ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΠΟΛΥΜΕΣΩΝ

Μελέτη στην τεχνολογία WebServices μέσω JavaScript

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΤΥΜΠΑΚΙΑΝΑΚΗ ΑΠΟΣΤΟΛΟΥ

Επιβλέπων Καθηγητής: Αθανάσιος Μαλάμος

Ηράκλειο, Απρίλιος 2010

(Υπογραφή)

.....

ΤΥΜΠΑΚΙΑΝΑΚΗΣ ΑΠΟΣΤΟΛΟΣ

Πτυχιούχος Τεχνολόγος Μηχανικός Εφαρμοσμένης Πληροφορικής και Πολυμέσων

Περίληψη

Σκοπός της πτυχιακής εργασίας ήταν η μελέτη και ανάπτυξη πρωτοτύπου στην τεχνολογία **AJAX** (**Asynchronous JavaScript And XML**) με τροφοδότηση μέσω υπηρεσίας διαδικτύου (**WebService**). Η τεχνολογία αυτή βρίσκεται σε πρώιμο στάδιο ανάπτυξης και παρουσιάζει εξαιρετικό πεδίο εφαρμογών στο διαδίκτυο. Το **AJAX** είναι ένα σύνολο τεχνικών που επιτρέπει την ασύγχρονη επικοινωνία μεταξύ μια διαδικτυακής εφαρμογής Πελάτη (**Client-side web application**) και του Διακομιστή (**Server-side**).

Στο πλαίσιο αυτής της τεχνολογίας κατασκευάστηκε μια διαδικτυακή εφαρμογή, η οποία αναλαμβάνει να αποθηκεύσει, να διαχειριστεί και να προβάλει ένα σύνολο επαφών του χρήστη. Οι επαφές αυτές βρίσκονται αποθηκευμένες στον Διακομιστή, ο οποίος και δέχεται αιτήσεις από την πλευρά του Πελάτη μέσω ασύγχρονων αιτημάτων (**requests**) **AJAX** για να εκτελέσει τις ενέργειες που επιθυμεί ο χρήστης.

Συγκεκριμένα, έγινε μελέτη της τεχνολογίας **AJAX**, δηλαδή του συνόλου των τεχνολογιών που την αποτελούν, και βασίζονται σε εφαρμογές στην πλευρά του Πελάτη (**Client-side**). Σε αυτήν την πλευρά χρησιμοποιήθηκαν οι τεχνολογίες **HTML**, **JavaScript** (**jQuery**) και **CSS**. Για την πλευρά της διαδικτυακής υπηρεσίας του Διακομιστή χρησιμοποιήθηκε η τεχνολογία **ColdFusion** της **Adobe**, καθώς και για τον ίδιο τον Διακομιστή που είναι ένας **ColdFusion Server**. Τέλος, για την αποθήκευση των επαφών χρησιμοποιήθηκε το πρότυπο **JSON**, το οποίο και είναι ένα εναλλακτικό πρότυπο της γλώσσας **XML** που αρχικά υποστήριζε το **AJAX**.

Η συγκεκριμένη διαδικτυακή εφαρμογή μπορεί να γίνει οδηγός για κατασκευή εφαρμογών με χρήση της τεχνολογίας **AJAX**, καθώς η όλη φιλοσοφία της έγκειται στην χρήση μίας μόνο σελίδας η οποία δεν χρειάζεται να ξαναφορτωθεί ποτέ αλλά μέσα από την οποία εκτελούνται όλες οι εργασίες της εφαρμογής με ασύγχρονη επικοινωνία στο παρασκήνιο, χωρίς να υπάρχει αρνητικό αντίκτυπο στον χρήστη με χαμένους χρόνους επαναφορτώσεων.

Λέξεις Κλειδιά: AJAX, Ασύγχρονη, jQuery, JavaScript, XML, ColdFusion, JSON, HTML

Abstract

The scope of this thesis was the study and development of a prototype based on the **AJAX** technology (**Asynchronous JavaScript and XML**), which feeds of a **WebService**. This technology is at an early stage of development and presents an exceptional range of applications on the Web. **AJAX** is a group of techniques which allows asynchronous communication between a web application on the Client-side and a Server.

A web application was build, based on **AJAX** technology, which undertakes the storage, administration and presentation of a sum of contacts of the user. These contacts are stored on the Server, and the Client-side application can then make asynchronous **AJAX** requests to the Server, giving the commands needed to be executed according to the user's actions.

More specifically, the group of technologies consisting **AJAX** were studied, based on applications on the client-side. On this side, the technologies of **HTML**, **JavaScript** (**jQuery** as the core framework) and **CSS** were studied and applied. On the server side, the technology of **ColdFusion** by **Adobe** was used as the web service, as well as for the Server itself, which is a **ColdFusion Server**. Finally, for the contacts storage, the **JSON** format was used, as an alternative to the **XML** language that the **AJAX** supported at its early stages.

This web application can be used as a guide for building **AJAX** applications, as the whole philosophy was the use of one single web page for everything. This results in a **Single-page application** which executes all its functions without reloading, making use of asynchronous communication at the background, saving the end user from waiting between long reloading times.

Keywords: AJAX, asynchronous, jQuery, JavaScript, XML, ColdFusion, JSON, HTML

Πίνακας περιεχομένων

1

Εισαγωγή

1.1 AJAX και Ασύγχρονη Επικοινωνία

Κοιτάζοντας πίσω μερικά χρόνια, η κατασκευή μιας διαδικτυακής εφαρμογής ήταν καταδικασμένη εξ' αρχής να υστερεί σε πολλά σημεία, έναντι των εφαρμογών που αναπτύσσονταν για "**offline**" (εκτός σύνδεσης) χρήση στους επιτραπέζιους υπολογιστές (**Desktop applications**). Η αφθονία λειτουργιών και η γενική ποιότητα αποκριτικότητας (**responsiveness**) των τελευταίων, έκαναν τις διαδικτυακές εφαρμογές ιδίας ποιότητας να φαντάζουν ένα μακρινό μέλλον.

Το πρόβλημα ήταν εμφανές. Σε πολλές περιπτώσεις, σχετικές σελίδες σε έναν ιστότοπο (**website**) αποτελούνταν από αρκετό περιεχόμενο το οποίο ήταν κοινό μεταξύ τους. Χρησιμοποιώντας παραδοσιακές μεθόδους, αυτό το περιεχόμενο έπρεπε να ξαναφορτωθεί σε κάθε αίτημα του Πελάτη. Αυτό το πρόβλημα ήρθε να λύσει το **AJAX.**

Με την χρήση του **AJAX**, μια διαδικτυακή εφαρμογή μπορεί να κάνει μια αίτηση προς τον Διακομιστή, ζητώντας μόνο το περιεχόμενο που χρειάζεται να ανανεωθεί, μειώνοντας έτσι δραστικά και τον όγκο των δεδομένων που πρέπει να μεταφερθούν, αλλά και τον χρόνο φόρτωσης του περιεχομένου και κατά συνέπεια τις σελίδας και γενικότερα της εφαρμογής.

Η χρήση ασύγχρονων αιτήσεων επιτρέπει στο Σύστημα Διεπαφής Χρήστη (**User Interface**) του περιηγητή Ιστού (**Web browser**) του Πελάτη να είναι πιο διαδραστικό (**interactive**) και να ανταποκρίνεται γρηγορότερα στις ενέργειες του χρήστη. Επιπλέον, συγκεκριμένα τμήματα

των σελίδων μπορούν να επαναφορτωθούν χωριστά, ατομικά. Οι τελικοί χρήστες εύκολα αντιλαμβάνονται την διαφορά, με την εφαρμογή να είναι πιο γρήγορη, να ανταποκρίνεται καλύτερα, ακόμα και όταν η τεχνολογία δεν έχει αλλάξει από την πλευρά του Διακομιστή.

Στα πλεονεκτήματα από την χρήση του **AJAX** έχουμε επίσης την μείωση των απαιτούμενων συνδέσεων προς των Διακομιστή, αφού μέρη του κώδικα (**scripts**) καθώς και τα φύλλα στυλ (**style sheets**) που απαιτούνται, χρειάζεται να αιτηθούν μία μόνο φορά. Τέλος, μπορεί να διατηρηθεί μια κατάσταση σε όλη την έκταση ενός ιστότοπου, αφού η μεταβλητές της **JavaScript** θα εξακολουθούν να υπάρχουν φορτωμένες, εφόσον η κεντρική σελίδα (**container page**) δεν χρειάζεται να ξαναφορτωθεί.

1.2 Αντικείμενο πτυχιακής

Σε αυτήν την πτυχιακή εργασία μελετάται το σύνολο των τεχνολογιών που απαρτίζουν το **AJAX**, και στην συνέχεια κατασκευάζεται μια διαδικτυακή εφαρμογή η οποία κάνει χρήση της τεχνολογίας αυτής για να παραχθεί μια λεγόμενη «**μονοσέλιδη εφαρμογή**» (**Single-page Application**). Η κατασκευή μια τέτοιας εφαρμογής είναι μια ριζοσπαστική αλλαγή από το κλασσικό μοτίβο του «**Αίτηση – Απάντηση**» που υπάρχει στην πλειοψηφία του διαδικτύου σήμερα. Δεν απαιτεί μόνο ένα εξαιρετικό στρώμα αφαίρεσης (**abstraction layer**) της **JavaScript** όπως το **jQuery**, αλλά καθιστά αναγκαία και μια πιο περίπλοκη αρχιτεκτονική συστήματος, όπως αυτή του **MVC** (**Model-View-Controller**).

Καθώς το **AJAX** ουσιαστικά αναφέρεται σε λύσεις για την πλευρά του Πελάτη, η χρήση μιας τεχνολογίας για την πλευρά του Διακομιστή είναι σχετικά ελεύθερη και όχι τόσο μεγάλης σημασίας. Άλλωστε, όπως αναφέρθηκε και προηγουμένως, οι εφαρμογές **AJAX** μπορούν να κατασκευαστούν και χωρίς να αλλάξει η πλευρά του Διακομιστή, κυρίως όσον αφορά την τεχνολογία και την αρχιτεκτονική της. Με αυτό το σκεπτικό επιλέχθηκε η τεχνολογία **ColdFusion** για την πλευρά του Διακομιστή, τόσο για τον ίδιο τον Διακομιστή που θα φιλοξενεί την εφαρμογή, όσο και για την διαδικτυακή υπηρεσία που κατασκευάστηκε και εκτελείται στην πλευρά του Διακομιστή για την εξυπηρέτηση της **AJAX** πλευράς του Πελάτη.

Στα πλαίσια λοιπόν της παραπάνω αρχιτεκτονικής, κατασκευάστηκε μια διαδικτυακή εφαρμογή, οι «**Έξυπνες Επαφές**», η οποία επιτρέπει στον τελικό χρήστη να αποθηκεύει, να επεξεργάζεται και να αναζητά μια λίστα επαφών, σαν ένα ηλεκτρονικό βιβλίο επαφών δηλαδή. Ο χρήστης μπορεί να δημιουργήσει μια νέα επαφή, να επεξεργαστεί μια ήδη υπάρχουσα, να αναζητήσει μια επαφή από την λίστα και να την προβάλει, και τέλος να διαγράψει μια επαφή. Όλες αυτές οι ενέργειες παρουσιάζονται και εκτελούνται σε ένα περιβάλλον μίας μόνο σελίδας, της οποίας το περιεχόμενο ο χρήστης βλέπει να αλλάζει ακόμα και ολοκληρωτικά, χωρίς όμως να ξαναφορτώνεται ποτέ η πραγματική σελίδα ή να

μεταφέρεται σε κάποια άλλη. Όλα γίνονται στο παρασκήνιο, με την εφαρμογή να επικοινωνεί μέσω ασύγχρονων αιτημάτων **AJAX** με τον Διακομιστή και να αλλάζει δυναμικά το περιεχόμενο της σελίδας ανάλογα με την απόκριση των αιτημάτων αυτών, δηλαδή των ενεργειών του χρήστη.

Η επιλογή ενός τέτοιου είδους εφαρμογής έγινε για διάφορους λόγους. Ο κύριος λόγος ήταν ότι η αρχιτεκτονική της «**μονοσέλιδης**» εφαρμογής είναι ένας αρκετά συμπαγής τρόπος κατασκευής μιας διαδικτυακής εφαρμογής, γεγονός το οποίο βοηθάει τον γενικότερο στόχο αυτής της πτυχιακής εργασίας, που δεν είναι άλλος από την επίδειξη των δυνατοτήτων της τεχνολογίας **AJAX**, δηλαδή την υψηλή ποιότητα της αμεσότητας, της λειτουργικότητας και της ευχρηστίας που λαμβάνει ο τελικός χρήστης μια τέτοιας εφαρμογής. Σήμερα, η γενική φιλοσοφία για την δημιουργία διαδικτυακών υπηρεσιών, επιβάλει την δημιουργία όλο και περισσότερο πλούσιων εφαρμογών, με μια τάση που θέλει τις καλύτερες πρακτικές που εφαρμόζονται στην πλευρά του Διακομιστή, να μεταφέρονται τώρα στον Πελάτη, απαλλάσσοντας τον Διακομιστή από αρκετό φόρτο και πολύτιμους πόρους. Καθώς η τεχνολογία προχωράει και η υπολογιστική ισχύ στην πλευρά του Πελάτη ολοένα και εμπλουτίζεται και γίνεται αρκετά ισχυρή, η παραπάνω φιλοσοφία και τάση είναι ακόμα ποιο ικανό και αναγκαίο να εφαρμοστεί. Η εφαρμογή «**Έξυπνες Επαφές**» ακολουθεί αυτήν την τάση και φιλοσοφία και την εκμεταλλεύεται στο έπακρο.

1.3 Οργάνωση κειμένου

Το **Κεφάλαιο 2** εκθέτει και περιγράφει αναλυτικά όλες τις τεχνολογίες που συνέβαλαν στην δημιουργία της εργασίας. Στο **Κεφάλαιο 3** γίνεται μια συνοπτική περιγραφή της αρχιτεκτονικής της εφαρμογής και περιγράφονται αναλυτικά όλες οι λειτουργίες της. Το **Κεφάλαιο 4** αναλύεται η σχεδίαση της εφαρμογής με λεπτομέρειες για την αρχιτεκτονική των δύο πλευρών που την αποτελούν. Στο **Κεφάλαιο 5** αναλύονται ορισμένα ενδεικτικά τεχνικά θέματα για την ανάπτυξη της εφαρμογής και δίνονται πληροφορίες για το λογισμικό που χρησιμοποιήθηκε και την διαδικασία εγκατάστασης της εφαρμογής. Το **Κεφάλαιο 6** με την βοήθεια ενός σεναρίου αξιολογεί την εφαρμογή και αποτελεί και το εγχειρίδιο χρήσης της. Στο **Κεφάλαιο 7** είναι ο επίλογος της εφαρμογής καθώς και μερικές ιδέες για μελλοντική επέκτασή της. Εφαρμογές σχετικές με το αντικείμενο της πτυχιακής εργασίας παρουσιάζονται στο **Κεφάλαιο 8 .** Το **Κεφάλαιο 9** αποτελεί την βιβλιογραφία της πτυχιακής εργασίας.

2

Θεωρητικό υπόβαθρο

Για την ανάπτυξη της εφαρμογής «**Έξυπνες Επαφές**» της πτυχιακής εργασίας, χρειάστηκε να μελετηθούν όλες οι τεχνολογίες και οι τεχνικές που αποτελούν το **AJAX**. Για να κατασκευαστεί μια ολοκληρωμένη εφαρμογή **AJAX**, ήταν απαραίτητο να αναπτυχθεί η τεχνολογία στην πλευρά του Διακομιστή αλλά και η τεχνολογία στην μεριά του Πελάτη. Οι τεχνολογίες που μελετήθηκαν και εφαρμόστηκαν είναι οι εξής:

Για την πλευρά του Πελάτη:

1. Γλώσσα προγραμματισμού **JavaScript**
2. Αντικείμενο **XMLHttpRequest**
3. Βιβλιοθήκη της **JavaScript**, **jQuery**
4. Γλώσσα Σήμανσης **HTML**(**Hypertext Markup Language**)
5. Γλώσσα φύλλων στυλ **Cascading Style Sheets** (**CSS**)

Για την πλευρά του Διακομιστή:

1. Γλώσσα Σήμανσης **ColdFusion** (**ColdFusion Markup Language**)
2. Τεχνολογία Διακομιστή **ColdFusion** (**ColdFusion Server**)
3. Μορφότυπο **JSON** (**JavaScript Object Notation data format**)

Για την γενικότερη ανάπτυξη της εφαρμογής, και κυρίως της αρχιτεκτονικής της:

1. Οι Υπηρεσίες Ιστού **Web Services**
2. Το Μοντέλο Αντικειμένου Εγγράφου (**Document Object Model**) ή **DOM**
3. Η αρχιτεκτονική λογισμικού **MVC** (**Model-View-Controller**)

Στις παρακάτω ενότητες θα δοθεί λεπτομερή περιγραφή, σε θεωρητική βάση, για τις προαναφερθείσες τεχνολογίες. Η γλώσσα **JavaScript** καθώς και η γλώσσες **HTML** και **CSS** θεωρούνται γνωστές και θα περιγραφούν συνοπτικά, ενώ επίσης συνοπτικά θα περιγραφεί και η γλώσσα **XML** η οποία ανήκει στις τεχνολογίες του αρχικού πυρήνα του **AJAX** αλλά δεν χρησιμοποιήθηκε σε αυτήν την εργασία.

2.1 AJAX (Asynchronous JavaScript and XML)

2.1.1 Τι είναι το AJAX;

Ο όρος «**AJAX**» αρχικά δημιουργήθηκε από την συντομογραφία του "**Asynchronous JavaScript and XML**". Ουσιαστικά το **AJAX** δεν είναι μια νέα τεχνολογία, ούτε μια νέα γλώσσα προγραμματισμού, αλλά ένας νέος τρόπος χρήσης των υπαρχόντων τεχνολογιών. Είναι ένα γκρουπ από αλληλένδετες τεχνικές ανάπτυξης ιστοσελίδων, που χρησιμοποιούνται στην πλευρά του Πελάτη (**client-side**) για την δημιουργία διαδραστικών διαδικτυακών εφαρμογών.

> *"**AJAX**: Η τέχνη της ανταλλαγής δεδομένων με ένα Διακομιστή διαδικτύου, και της αλλαγής μερών μιας ιστοσελίδας, χωρίς να ξαναφορτωθεί ολόκληρη η ιστοσελίδα."*
>
> ***-- w3schools.com***

Με το **AJAX**, οι διαδικτυακές εφαρμογές μπορούν να ανακτούν δεδομένα από τον Διακομιστή (**server**) ασύγχρονα, δηλαδή στο παρασκήνιο, χωρίς να παρεμβαίνουν στην εμφάνιση και την συμπεριφορά της υπάρχουσας σελίδας. Η χρήση των τεχνικών του **AJAX** έχει οδηγήσει σε αύξηση των διαδραστικών ή δυναμικών διεπαφών στις ιστοσελίδες. Συνήθως, τα δεδομένα ανακτώνται χρησιμοποιώντας το αντικείμενο (**object**) **XMLHttpRequest**, το οποίο θα αναλυθεί παρακάτω.

Σε αυτό το σημείο, αξίζει να επισημανθεί το εξής οξύμωρο. Παρά το όνομά του, το **AJAX** δεν απαιτεί την χρήση της **XML** για την λήψη των δεδομένων, καθώς επίσης δεν είναι απαραίτητο τα αιτήματα προς των Διακομιστή να είναι ασύγχρονα. Αυτό συμβαίνει γιατί από το 2005, που αρχικά παρουσιάστηκε το **AJAX** ως ιδέα, μέχρι σήμερα, οι προγραμματιστές διαδικτύου το έχουν εμπλουτίσει με διάφορους τρόπους και το έχουν επεκτείνει ενσωματώνοντας αρκετές νέες τεχνολογίες. Το όνομα όμως έχει παραμείνει το ίδιο, καθώς ήταν απαραίτητο ένα διακριτό αναγνωριστικό για την χρήση όλων αυτών των τεχνολογιών, κάτω από ένα κοινό όρο ώστε να μην υπάρχει σύγχυση.

2.1.2 Ιστορία

Η φόρτωση ασύγχρονου περιεχομένου έγινε πραγματικότητα για πρώτη φορά το 1995, με την πρώτη έκδοση της γλώσσας προγραμματισμού **Java**. Τότε παρουσιάστηκαν για πρώτη φορά τα «**Java Applets**», τα οποία επέτρεπαν «**μεταγλωττισμένο**» (**compiled**) κώδικα στην μεριά του Πελάτη (**client-side**) να φορτώσει δεδομένα ασύγχρονα από τον διαδικτυακό Διακομιστή (**web server**), μετά την φόρτωση μιας ιστοσελίδας. Την ίδια εφαρμογή είχε και το «**IFrame**» στοιχείο της γλώσσας σήμανσης **HTML**, το οποίο παρουσίασε μια χρονιά αργότερα, το 1996, ο **Internet Explorer** της **Microsoft**. Το 1999 έγινε το μεγάλο βήμα, με τον **Internet Explorer 5** να ενσωματώνει στα **ActiveX Controls** του το «**περιτύλιγμα**» (**wrapper**) **XMLHTTP**, το οποίο μέχρι και σήμερα χρησιμοποιείται από σχεδόν όλους τους περιηγητές Διαδικτύου (**Internet Browsers**) ως το έμφυτο (**native**) αντικείμενο **XMLHttpRequest**.

Όμως, παρά την κοινή αποδοχή και την αναγνώριση της σπουδαιότητας αυτής της τεχνολογίας, έπρεπε να περάσουν αρκετά χρόνια, ώσπου το 2004 πρώτη η **Google**, να κάνει εκτεταμένη χρήση του **AJAX** στο **Gmail**, μια υπηρεσία ανταλλαγής ηλεκτρονικής αλληλογραφίας. Μέχρι τότε η χρησιμότητα των **HTTP** αιτήσεων προς τον Διακομιστή στο παρασκήνιο, καθώς και γενικά οι ασύγχρονες τεχνολογίες στο διαδίκτυο, παρέμεναν αρκετά ασαφής και η εφαρμογή και υποστήριξη τους από τους προγραμματιστές ήταν πολύ περιορισμένη. Το 2005, η **Google** και πάλι, με το **Google Maps**, μια υπηρεσία περιήγησης χαρτών, έκανε ευρέως γνωστό το **AJAX**, και πολλοί ερευνητές και προγραμματιστές άρχισαν να ασχολούνται μαζί του, παρόλο που δεν υπήρχε καν σαν όρος ή ορισμός τότε.

> *Ερ.: Γιατί ένιωσες την ανάγκη να δώσεις ένα όνομα σ' αυτό;*
>
> *Απ.: Χρειαζόμουν κάτι συντομότερο από το «Ασύγχρονη **JavaScript** + **CSS** + **DOM** + **XMLHttpRequest**" για να χρησιμοποιώ όταν συζητούσα αυτήν την προσέγγιση με τους πελάτες.*
>
> ***-- Jesse James Garrett***

"Νονός" του όρου «**AJAX**» είναι ο **Jesse James Garrett**, Πρόεδρος και ιδρυτής της εταιρίας **Adaptive Path**, ο οποίος πρώτος στον κόσμο, στο άρθρο του "**Ajax: A New Approach to Web Applications**" τον Φεβρουάριο του 2005, έδωσε ένα όνομα στην δυνατότητα που υπήρχε με τις τότε υπάρχουσες τεχνολογίες, να γεφυρωθεί το κενό μεταξύ Διακομιστή (**Server**) και Πελάτη (**Client**) με την ασύγχρονη επικοινωνία μεταξύ αυτών.

Στις 5 Απριλίου του 2006, η διεθνής κοινοπραξία για την κατοχύρωση προτύπων του παγκόσμιου ιστού **WC3** (**World Wide Web Consortium**) ανακοίνωσε το πρώτο προσχέδιο

προδιαγραφής για το αντικείμενο, σε μια προσπάθεια να δημιουργήσει ένα επίσημο διαδικτυακό πρότυπο.

2.1.3 Τεχνολογίες

Ο όρος «**AJAX**» ήρθε να αναπαραστήσει ένα ευρύ γκρουπ από διαδικτυακές τεχνολογίες οι οποίες μπορούν να χρησιμοποιηθούν για να παρέχουν τα μέσα σε μια διαδικτυακή εφαρμογή να επικοινωνήσει με έναν Διακομιστή στο παρασκήνιο, χωρίς να παρεμβαίνουν με την τρέχουσα κατάσταση της σελίδας. Στο άρθρο του όπου επινόησε τον όρο «**AJAX**», ο **Jesse James Garrett** εξηγούσε ότι οι απαιτούμενες τεχνολογίες ήταν οι εξής:

- Την γλώσσα σήμανσης **HTML** ή την **XHTML** και την γλώσσα φύλλων στυλ **CSS**, για την παρουσίαση
- Το Μοντέλο Αντικειμένου Εγγράφου (**Document Object Model**) ή **DOM**, για την δυναμική έκθεση των δεδομένων καθώς και την αλληλεπίδραση με αυτά.
- Την γλώσσα σήμανσης **XML** και την γλώσσα μετασχηματισμού **XSLT** για την ανταλλαγή, την διαχείριση και προβολή των δεδομένων
- Το αντικείμενο **XMLHttpRequest** για την ασύγχρονη επικοινωνία
- Την γλώσσα προγραμματισμού **JavaScript** σαν ενωτικό κρίκο όλων των παραπάνω τεχνολογιών

Έκτοτε όμως, επήλθαν πολλές εξελίξεις στις τεχνολογίες που χρησιμοποιούνταν για την ανάπτυξη μιας εφαρμογής **AJAX**, καθώς και στον καθορισμό του όρου **AJAX.** Ειδικότερα, έχει διαπιστωθεί ότι:

- Η γλώσσα **JavaScript** δεν είναι η μόνη γλώσσα προγραμματισμού από την πλευρά του Πελάτη που μπορεί να χρησιμοποιηθεί για την ανάπτυξη μιας εφαρμογής **AJAX**. Άλλες γλώσσες όπως η **VBScript** είναι ικανές να παρέχουν την απαιτούμενη λειτουργικότητα. Παρόλα αυτά όμως, η **JavaScript** έχει παραμείνει η πιο δημοφιλής γλώσσα στον προγραμματισμό του **AJAX** κυρίως για την ενσωμάτωση και την συμβατότητα που έχει με την πλειοψηφία των σύγχρονων περιηγητών Ιστού (**Web Browsers**).
- Η γλώσσα σήμανσης **XML** δεν είναι απαραίτητη για την ανταλλαγή δεδομένων και κατά συνέπεια και η γλώσσα μετασχηματισμού **XSLT** δεν είναι αναγκαία για την διαχείριση των δεδομένων. Συχνά χρησιμοποιείται το μορφότυπο **JSON** (**JavaScript Object Notation**) σαν ένα εναλλακτικό πρότυπο για την ανταλλαγή δεδομένων, παρόλο που και άλλα πρότυπα, όπως η προ-διαμορφωμένη (**preformatted**) **HTML** ή ακόμα και το απλό κείμενο, μπορούν επίσης να χρησιμοποιηθούν.

2.1.4 Μειονεκτήματα και Αδυναμίες

Λόγω της δυναμικής φύσης τους, οι **AJAX** διεπαφές είναι συχνά δυσκολότερο να αναπτυχθούν σε σύγκριση με τις στατικές σελίδες. Οι εφαρμογές **AJAX** αναπόφευκτα περιλαμβάνουν εκτέλεση πολύπλοκων τμημάτων **JavaScript** κώδικα στην μεριά του Πελάτη. Το να γίνει ένας τέτοιος περίπλοκος κώδικας αποδοτικός και χωρίς σφάλματα, είναι ένας στόχος που πρέπει να ληφθεί σοβαρά υπόψη.

Επίσης, οι σελίδες που δημιουργήθηκαν δυναμικά με την χρήση διαδοχικών αιτημάτων **AJAX** δεν καταγράφουν αυτόματα τον εαυτό τους στην μηχανή ιστορικού του Περιηγητή (**browser**). Αυτό έχει ως αποτέλεσμα το πάτημα του κουμπιού «**Πίσω**» του Περιηγητή να μην μεταφέρει τον χρήστη πίσω σε μια προγενέστερη κατάσταση της **AJAX** σελίδας, αλλά πιθανόν να τον μεταφέρει στην τελευταία ολόκληρη σελίδα που είχε επισκεφτεί πριν από αυτή. Κάποιες τεχνικές επίλυσης αυτού του προβλήματος περιλαμβάνουν την χρήση αόρατων «**IFrames**» για να προκαλέσουν αλλαγές στο ιστορικό του Περιηγητή και για να αλλάζουν το «**μέρος άγκυρας**» (**anchor portion**) της διεύθυνσης **URL** (ακολουθούμενο από ένα σύμβολο «#» ή **hash**) όταν εκτελεστεί το **AJAX**, και παρακολουθώντας το στην συνέχεια για αλλαγές.

Οι αλλεπάλληλες ενημερώσεις των δυναμικών σελίδων έχουν δημιουργήσει ένα επιπλέον πρόβλημα, που έγκειται στο ότι ο χρήστης δύσκολα μπορεί να προσθέσει στους σελιδοδείκτες του μια συγκεκριμένη κατάσταση της εφαρμογής. Λύσεις για αυτό το πρόβλημα υπάρχουν, και πολλές από αυτές βασίζονται στην χρήση του «**αναγνωριστικού κομματιού**» (**fragment identifier**) της διεύθυνσης **URL** (το κομμάτι ενός **URL** που βρίσκεται μετά το σύμβολο «#») για να παρακολουθούν, και επομένως να επιτρέπουν στον χρήστη να αποθηκεύει, μια δεδομένη κατάσταση της εφαρμογής.

Είναι γεγονός ότι οι περισσότεροι αυτόματοι καταχωρητές διαδικτύου (**web crawlers**) δεν εκτελούν κώδικα **JavaScript**. Επομένως, οι διαδικτυακές εφαρμογές που έχουν προοριστεί για δημόσια (ανοικτή προς όλους) καταχώρηση, πρέπει να παρέχουν έναν εναλλακτικό μέσο για την προσπέλαση του περιεχομένου τους το οποίο, υπό κανονικές συνθήκες, θα είχε ανακτηθεί με την χρήση του **AJAX**, έτσι ώστε να επιτρέπουν στις μηχανές αναζήτησης να το καταχωρήσουν στις βάσεις τους.

Ένα πολύ σημαντικό μειονέκτημα είναι αυτό της συμβατότητας με συσκευές που δεν υποστηρίζουν όλες τις τεχνολογίες που αξιοποιεί μια **AJAX** εφαρμογή. Κάθε χρήστης που ο Περιηγητής του δεν υποστηρίζει την χρήση της JavaScript ή του αντικειμένου **XMLHttpRequest**, ή απλώς έχει απενεργοποιημένες αυτές τις λειτουργίες, δεν θα είναι σε θέση να χρησιμοποιήσει κατάλληλα τις σελίδες που στηρίζονται στην χρήση του **AJAX**. Ομοίως, συσκευές όπως κινητά τηλέφωνα, προσωπικοί ηλεκτρονικοί βοηθοί (**Personal**

Digital Assistants ή **PDAs**) και οθόνες ανάγνωσης (**Screen Readers**) ενδεχομένως να μην παρέχουν υποστήριξη για τις απαιτούμενες τεχνολογίες. Ακόμη και οι οθόνες ανάγνωσης που υποστηρίζουν **AJAX**, μπορεί να μην έχουν την δυνατότητα να «διαβάσουν» κατάλληλα το δυναμικά παραγόμενο περιεχόμενο. Ο μόνος τρόπος για να επιτραπεί στον χρήστη να χρησιμοποιήσει την εφαρμογή, είναι να γυρίσουμε πίσω σε μεθόδους που δεν χρησιμοποιούν καθόλου την **JavaScript**. Αυτό μπορεί να επιτευχθεί μόνο εφόσον επιβεβαιωθεί ότι όλες οι διασυνδέσεις (**links**) και οι φόρμες (**forms**) της εφαρμογής μπορούν να αποδοθούν κατάλληλα, και δεν θα βασίζονται αποκλειστικά στην χρήση **AJAX.** Έπειτα, από την πλευρά της **JavaScript**, η υποβολή της φόρμας θα μπορούσε να σταματήσει με την εντολή «**return false**».

Ένας πολύ σημαντικός περιορισμός στην χρήση του **AJAX** έγκειται στην κοινή πολιτική καταγωγής (**Same Origin Policy**). Η πολιτική αυτή είναι ένα σημαντικό σχέδιο ασφάλειας το οποίο ισχύει για ορισμένες γλώσσες προγραμματισμού που εκτελούνται στην πλευρά του Περιηγητή (δηλαδή του Πελάτη) όπως η **JavaScript**. Συνοπτικά, η πολιτική αυτή επιτρέπει στα τρέχοντα κομμάτια κώδικα (**running scripts**) των σελίδων, τα οποία προέρχονται από τον ίδιο δικτυακό τόπο, να έχουν πρόσβαση στις μεταξύ τους μεθόδους και ιδιότητες χωρίς ειδικούς περιορισμούς, αλλά τους αποτρέπει να έχουν πρόσβαση στο μεγαλύτερο μέρος των μεθόδων και των ιδιοτήτων κατά μήκος των σελίδων που βρίσκονται σε διαφορετικούς δικτυακούς τόπους. Με βάση τα παραπάνω λοιπόν, μερικές τεχνικές που εφαρμόζονται στο **AJAX** δεν επιτρέπεται να χρησιμοποιηθούν κατά μήκος των τομέων (**domains**), αλλά θα πρέπει να περιοριστούν σε χρήση μόνο κάτω από τον ίδιο τομέα (**domain**)**.** Μια λύση έρχεται από την κοινοπραξία **W3C**, με την διάθεση ενός προσχεδίου πάνω στο αντικείμενο **XMLHttpRequest** το οποίο προορίζεται να του επιτρέψει αυτήν την λειτουργία.

Τελειώνοντας, πρέπει να σημειωθεί πως όπως και άλλες τεχνολογίες του διαδικτύου, το **AJAX** έχει το δικό του σύνολο από τρωτά σημεία, τα οποία οι προγραμματιστές καλούνται να αντιμετωπίσουν. Είναι γεγονός, ότι προγραμματιστές με εξοικείωση σε άλλες τεχνολογίες του διαδικτύου θα πρέπει να μελετήσουν πάνω σε νέες μεθόδους κωδικοποίησης και δοκιμών, για να είναι σε θέση να γράψουν ασφαλείς **AJAX** εφαρμογές. Ένα επιπλέον θέμα που προκύπτει αφορά το ότι οι Διεπαφές που κάνουν χρήση του **AJAX** μπορούν να αυξήσουν δραματικά τον αριθμό των αιτήσεων που παράγουν οι χρήστες, προς τους Διακομιστές διαδικτύου καθώς και στις λειτουργίες που κρύβουν πίσω τους, όπως οι Βάσεις Δεδομένων και άλλες. Αυτό μοιραία μπορεί να οδηγήσει σε μεγαλύτερους χρόνους απόκρισης ή/και πρόσθετες ανάγκες για επιπλέον ή καλύτερο υλικό (**hardware**).

2.2 Η Γλώσσα Σήμανσης HTML

Η γλώσσα **HTML**, της οποίας η ονομασία προέρχεται από τα αρχικά του «**HyperText Markup Language**» («**Γλώσσα Σήμανσης Υπερκειμένου**»), είναι η κυρίαρχη γλώσσα σήμανσης για τις ιστοσελίδες. Παρέχει τα μέσα για την δημιουργία δομημένων εγγράφων (**structured documents**) με την υποδήλωση μιας δομικής σημασιολογίας σε κείμενο όπως επικεφαλίδες, παράγραφοι, λίστες κ.λπ., καθώς και σε συνδέσεις (**links**), παραθέσεις (**quotes**) και άλλα στοιχεία. Επιτρέπει σε εικόνες και αντικείμενα να ενσωματωθούν και να χρησιμοποιηθούν για την δημιουργία διαδραστικών φορμών (**interactive forms**). Είναι γραμμένη σε μορφή «στοιχείων **HTML**» που αποτελούνται από «ετικέτες» (**tags**) οι οποίες περιβάλλονται από γωνιακές αγκύλες («<» και «>») και βρίσκονται εντός του περιεχομένου της ιστοσελίδας. Μπορεί να συμπεριλαμβάνει (**include**) ή να φορτώνει κομμάτια κώδικα (**scripts**) σε γλώσσες όπως η **JavaScript** τα οποία επιδρούν την συμπεριφορά των επεξεργαστών της **HTML**, όπως οι Περιηγητές Ιστού (**Web Browsers**) και τα φύλλα στυλ **CSS**, για τον καθορισμό της εμφάνισης και της διάταξης του κειμένου και άλλου υλικού. Η κοινοπραξία **W3C**, η οποία συντηρεί τα πρότυπα της **HTML** και των **CSS**, ενθαρρύνει την χρήση των φύλλων στυλ **CSS** πάνω σε σαφή παρουσιαστική σήμανση.

2.3 Η γλώσσα φύλλων στυλ CSS

Η **Cascading Style Sheets** ή **CSS** είναι μια γλώσσα φύλλων στυλ, η οποία χρησιμοποιείται για να περιγράψει την σημασιολογία παρουσίασης (δηλαδή την όψη και την μορφοποίηση) ενός εγγράφου που είναι γραμμένο σε μια γλώσσα σήμανσης. Η πιο κοινή εφαρμογή της είναι η περιγραφή του στυλ ιστοσελίδων που είναι γραμμένες σε γλώσσα **HTML** ή **XHTML**, αλλά μπορεί επίσης να εφαρμοστεί σε οποιοδήποτε είδους έγγραφο γραμμένο σε κώδικα **XML**. Η **CSS** είναι κυρίως σχεδιασμένη για να επιτρέπει τον διαχωρισμό του περιεχομένου του εγγράφου (γραμμένο σε **HTML** ή κάποια παρόμοια γλώσσα σήμανσης) από την παρουσίαση του εγγράφου, συμπεριλαμβανομένων των στοιχείων όπως είναι η διάταξη, τα χρώματα και οι γραμματοσειρές. Αυτός ο διαχωρισμός μπορεί να βελτιώσει την προσβασιμότητα του περιεχομένου, να παρέχει περισσότερη ευελιξία και έλεγχο στον προσδιορισμό των χαρακτηριστικών της παρουσίασης, να επιτρέπει σε πολλαπλές σελίδες να μοιράζονται την ίδια μορφοποίηση, και να μειώσει την πολυπλοκότητα και την επανάληψη του δομικού περιεχομένου (π.χ. το ότι επιτρέπει τον σχεδιασμό ιστοσελίδων χωρίς την χρήση πινάκων) . Η **CSS** μπορεί επίσης να επιτρέψει στην ίδια σελίδα σήμανσης να παρουσιαστεί με διαφορετικά στυλ για διαφορετικές μεθόδους απόδοσης, όπως είναι η απόδοση σε οθόνη, σε εκτύπωση, σε φωνή (όταν διαβάζεται από ένα πρόγραμμα περιήγησης βασισμένο στην

ομιλία) και σε συσκευές αφής για ανθρώπους με προβλήματα όρασης βασισμένες στο σύστημα **Braille.** Ενώ ο συντάκτης ενός εγγράφου μπορεί τυπικά να συνδέσει το έγγραφο με ένα φύλλο στυλ **CSS**, ο αναγνώστης μπορεί να χρησιμοποιήσει ένα διαφορετικό φύλλο στυλ, ίσως κάποιο από τον δικό του υπολογιστή, για να παρακάμψει αυτό που ο συντάκτης έχει ορίσει. Η **CSS** προσδιορίζει λεπτομερώς ένα σύστημα προτεραιότητας για τον καθορισμό των κανόνων στυλ που πρέπει να εφαρμοστούν, εάν για ένα συγκεκριμένο στοιχείο αντιστοιχούν περισσότεροι από ένας κανόνες. Οι προδιαγραφές για την γλώσσα **CSS** τηρούνται από την κοινοπραξία **W3C**. Ως «τύπος διαδικτυακού μέσου» (**Internet media type**) για την **CSS** έχει καταχωρηθεί να χρησιμοποιείται το «**text/css**».

2.4 Η γλώσσα JavaScript

Η **JavaScript** είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού που ανήκει στην κατηγορία των γλωσσών «**σεναρίου**» («**scripting languages**»), δηλαδή των γλωσσών που επιτρέπουν τον έλεγχο μίας ή περισσοτέρων εφαρμογών λογισμικού. Συνήθως οι «**scripting**» γλώσσες είναι ενσωματωμένες εντός της εφαρμογής λογισμικού που ελέγχουν. Η **JavaScript** επιτρέπει την προγραμματιστική πρόσβαση σε αντικείμενα που περιέχονται σε μία εφαρμογή στην πλευρά του Πελάτη και σε άλλου είδους εφαρμογών. Χρησιμοποιείται κατά κύριο λόγο με την μορφή της «**JavaScript** στην πλευρά του Πελάτη» («**client-side JavaScript**»), η οποία υλοποιείται ως ένα ενσωματωμένο εξάρτημα (**integrated component**) του προγράμματος περιήγησης, επιτρέποντας την ανάπτυξη εμπλουτισμένων διεπαφών χρήστη και δυναμικών ιστοσελίδων.

Η **JavaScript** είναι μια διάλεκτος που ακολουθεί το πρότυπο «**ECMAScript**» και χαρακτηρίζεται ως μία δυναμική, «χαλαρής δακτυλογράφησης» («**loose typing**» ή «**weak typing**»), βασισμένης σε πρωτότυπο («**prototype-based**») γλώσσα με πρώτης κλάσης συναρτήσεις. Η **JavaScript** επηρεάστηκε από πολλές γλώσσες και σχεδιάστηκε να μοιάζει με την **Java**, αλλά να είναι ευκολότερη στο να «δουλευτεί» από μη προγραμματιστές. Η **JavaScript**, παρά το όνομα, δεν έχει ουσιαστικά καμία σχέση με την γλώσσα **Java**, παρά το γεγονός ότι έχουν επιφανειακές ομοιότητες μεταξύ τους.

2.5 Το Μοντέλο DOM (Document Object Model)

2.5.1 Τι είναι το DOM;

Το Μοντέλο Αντικειμένου Εγγράφου (**DOM**) είναι μια σύμβαση για τον συμβολισμό και την αλληλεπίδραση με τα αντικείμενα (**objects**) μέσα σε ένα έγγραφο **HTML**, **XHTML** και **XML**, η οποία είναι ανεξάρτητη πλατφόρμας (**cross-platform**) και ανεξάρτητη γλώσσας προγραμματισμού (**language-independent**). Τα συστατικά του μοντέλου **DOM**, όπως για παράδειγμα τα «Στοιχεία» του («**Elements**»), μπορούν να «διευθυνσιοδοτηθούν» και να διαχειριστούν μέσα από το συντακτικό της γλώσσας προγραμματισμού που χρησιμοποιείται. Η δημόσια διεπαφή ενός μοντέλου **DOM** καθορίζεται μέσα από στην δική του Διεπαφή Ανάπτυξης Εφαρμογών (**Application Programming Interface** ή **API**).

2.5.2 Ιστορία

Η ιστορία του μοντέλου **DOM** είναι συνυφασμένη με την ιστορία του λεγόμενου και «Πόλεμου των Περιηγητών» στα τέλη της δεκαετίας του 1990 μεταξύ του **Netscape Navigator** και του **Microsoft Internet Explorer**, καθώς και με εκείνη της γλώσσας **JavaScript** και της γλώσσας **JScript**, οι οποίες ήταν οι πρώτες γλώσσες προγραμματισμού που ενσωματώθηκαν ευρέως στις «μηχανές διάταξης» (**layout engines**) των περιηγητών Ιστού.

2.5.2.1 DOM Επιπέδου 0

Η γλώσσα **JavaScript** κυκλοφόρησε από την **Netscape Communications** το 1996 εντός του περιηγητή της **Netscape Navigator 2.0**. Η ανταγωνίστρια της **Netscape**, η **Microsoft**, κυκλοφόρησε τον περιηγητή **Internet Explorer 3.0** αργότερα την ίδια χρονιά με μία «μεταφορά» (**port**) της γλώσσας **JavaScript**, με το όνομα **JScript**. Και οι δυο γλώσσες επέτρεπαν στους προγραμματιστές διαδικτύου να δημιουργούν ιστοσελίδες με διαδραστικότητα στην πλευρά του Πελάτη (**client-side interactivity**). Οι περιορισμένες δυνατότητες της πρώτης γενιάς αυτών των γλωσσών, για τον εντοπισμό των συμβάντων που είχαν προκληθεί από τον χρήστη (**user-generated events**) και της τροποποίησης του εγγράφου **HTML**, έγιναν τελικά γνωστές ως «**DOM Επιπέδου 0**» ή «**Legacy DOM**» («κληρονομιά **DOM**»). Κανένα ανεξάρτητο πρότυπο δεν αναπτύχθηκε για το «**DOM Επιπέδου 0**», αλλά περιγράφτηκε εν μέρει στις προδιαγραφές της γλώσσας **HTML4**.

Το «**Legacy DOM**» είχε περιορισμούς στα είδη των στοιχείων (**elements**) που έδινε πρόσβαση. Τα στοιχεία που αντιπροσώπευαν φόρμα, σύνδεση (**link**) και εικόνα μπορούσαν να αναφερθούν με μία ιεραρχική ονομασία που ξεκινούσε από το αντικείμενο «ρίζα» (**root**

element) του εγγράφου. Μια ιεραρχική ονομασία μπορούσε να κάνει χρήση είτε των ονομάτων είτε του διαδοχικού δείκτη (**sequential index**) των στοιχείων που διέσχιζε. Για παράδειγμα, ένα στοιχείο φόρμας εισόδου (**form input element**) μπορούσε να προσπελαθεί είτε ως «έγγραφο.όνομαΦόρμας.όνομαΕισόδου» είτε ως «έγγραφο.φόρμες[0].στοιχεία[0]». Ένα σημαντικό χαρακτηριστικό του «**Legacy DOM**» ήταν η δυνατότητα να γίνει έλεγχος εγκυρότητας μια φόρμας από την πλευρά του Πελάτη, με την χρήση του δημοφιλούς εφέ «**rollover**» («ανατροπής»).

2.5.2.2 *Ενδιάμεσο DOM*

Το 1997, η **Netscape** και η **Microsoft** κυκλοφόρησαν αμφότερες τις εκδόσεις **4.0** των **Netscape Navigator** και **Internet Explorer** αντίστοιχα, προσθέτοντας υποστήριξη για την **Dynamic HTML** (**DHTML**), λειτουργία που επιτρέπει αλλαγές σε ένα φορτωμένο έγγραφο **HTML**. Η **DHTML** απαιτούσε «επεκτάσεις» (**extensions**) στο στοιχειώδες αντικείμενο εγγράφου που ήταν διαθέσιμο στις υλοποιήσεις του «**Legacy DOM**». Αν και οι υλοποιήσεις του «**Legacy DOM**» ήταν σε μεγάλο βαθμό συμβατές, επειδή η **JScript** ήταν βασισμένη στην **JavaScript**, οι «επεκτάσεις» της **DHTML** στο μοντέλο **DOM** είχαν αναπτυχθεί παράλληλα και από τους δυο κατασκευαστές των περιηγητών και παρέμειναν μη συμβατές. Αυτές οι εκδόσεις του μοντέλου **DOM** έμειναν γνωστές ως το «**Intermediate DOM**» (Ενδιάμεσο **DOM**).

Οι εκδόσεις των «**Intermediate DOM**» επέτρεπαν τον χειρισμό των ιδιοτήτων (**properties**) των φύλλων στυλ **Cascading Style Sheets** (**CSS**), οι οποίες επηρέαζαν την εμφάνιση ενός εγγράφου. Έδιναν επίσης πρόσβαση σε ένα νέο χαρακτηριστικό με το όνομα «**layers**» («στρώματα») μέσω της ιδιότητας «**document.layers**» (στον **Netscape Navigator**) και της ιδιότητας «**document.all**» (για τον **Internet Explorer**). Εξ' αιτίας των θεμελιωδών ασυμβατοτήτων ανάμεσα στα δυο «**Intermediate DOM**», η «ανεξάρτητη-περιηγητή» («**cross-browser**») ανάπτυξη απαιτούσε ειδικό χειρισμό για κάθε υποστηριζόμενο πρόγραμμα περιήγησης. Μεταγενέστερες εκδόσεις του **Netscape Navigator** εγκατέλειψαν την υποστήριξη για τον δικό του μοντέλο «**Intermediate DOM**». Ο **Internet Explorer** συνεχίζει να υποστηρίζει το δικό του μοντέλο για λόγους συμβατότητας «προς τα πίσω» (**backwards compatibility**).

2.5.2.3 Τυποποίηση

Η κοινοπραξία **World Wide Web Consortium** (**W3C**), ιδρύθηκε το 1994 για την προώθηση ανοικτών προτύπων (**open standards**) για τον Παγκόσμιο Διαδικτυακό Ιστό. Ήταν αυτή που μάζεψε στο ίδιο τραπέζι την **Netscape Communications**, την **Microsoft** μαζί με άλλες εταιρίες για την ανάπτυξη ενός προτύπου για τις γλώσσες προγραμματισμού των περιηγητών,

το οποίο ονομάστηκε «**ECMAScript**». Η πρώτη έκδοση του προτύπου δημοσιεύτηκε το 1997. Ακολούθησαν εκδόσεις της γλώσσας **JavaScript** και της **JScript** οι οποίες ενσωμάτωσαν το πρότυπο «**ECMAScript**» για μεγαλύτερη συμβατότητα, ανεξάρτητα του προγράμματος περιήγησης.

Μετά την κυκλοφορία του προτύπου «**ECMAScript**», η **W3C** άρχισε τις εργασίες για ένα τυποποιημένο μοντέλο **DOM**. Το αρχικό πρότυπο μοντέλου **DOM**, γνωστό ως «**DOM Επίπεδου 1**», προτάθηκε από την **W3C** στα τέλη του 1998. Σχεδόν ταυτόχρονα κυκλοφόρησε η έκδοση **5.0** του **Internet Explorer** η οποία παρείχε περιορισμένη υποστήριξη για το μοντέλο «**DOM Επιπέδου 1**». Το «**Επίπεδο 1**» παρείχε ένα πλήρες μοντέλο για ένα ολόκληρο έγγραφο **HTML** ή **XML**, συμπεριλαμβανομένων και των μέσων για την αλλαγή οποιουδήποτε μέρους του εγγράφου αυτού. Μη συμβατά προγράμματα περιήγησης, όπως ο **Internet Explorer 4.x** και **ο Netscape Navigator 4.x**, εξακολουθούσαν να χρησιμοποιούνται ευρέως ως τα τέλη του 2000.

Στα τέλη του 2000 δόθηκε στην δημοσιότητα το πρότυπο «**DOM Επιπέδου 2**». Εισήγαγε την συνάρτηση (**function**) «**getElementById**» καθώς και ένα μοντέλο συμβάντων (**event model**), ενώ παρείχε και υποστήριξη για τα «**namespaces**» της **XML** και την γλώσσα φύλλων στυλ **CSS**. Το πρότυπο «**DOM Επιπέδου 3**», που αποτελεί και την τρέχουσα έκδοση της προδιαγραφής για το μοντέλο **DOM**, δημοσιεύτηκε τον Απρίλιο του 2004, και προσθέτει υποστήριξη για την γλώσσα «**XPath**» καθώς και χειρισμό συμβάντων του πληκτρολογίου. Επίσης προστέθηκε μια διεπαφή για την «σειριοποίηση» (**serialization**) εγγράφων ως κώδικα **XML**.

Μέχρι το 2005, μεγάλα τμήματα του προτύπου **DOM** της **W3C** απολάμβαναν πολύ καλής υποστήριξης από τα κοινά προγράμματα περιήγησης που ήταν «συμμορφωμένα» με το πρότυπο «**ECMAScript**», συμπεριλαμβανομένων των **Internet Explorer 6** (2001), **Opera**, **Safari** και όλων των περιηγητών που στηρίζονται στην μηχανή **Gecko** της **Mozilla** (όπως ο **Firefox** και ο **Camino**).

2.5.3 Εφαρμογές στα προγράμματα Περιήγησης Ιστού

Ένας περιηγητής Ιστού δεν είναι υποχρεωμένος να χρησιμοποιεί το μοντέλο **DOM** προκειμένου να απεικονίσει (**render**) ένα έγγραφο **HTML**. Ωστόσο, το μοντέλο **DOM** απαιτείται από την γλώσσα **JavaScript** όταν πρόκειται να επιθεωρήσει (**inspect**), ή να τροποποιήσει, δυναμικά μια ιστοσελίδα. Με άλλα λόγια, το «**Document Object Model**» είναι ο τρόπος που η **JavaScript** βλέπει την περιεχόμενη **HTML** σελίδα της και την κατάσταση (**state**) του προγράμματος περιήγησης.

2.5.4 Υλοποιήσεις

Δεδομένου ότι το μοντέλο **DOM** υποστηρίζει πλοήγηση σε οποιαδήποτε κατεύθυνση (π.χ., του «πατέρα» και του προηγούμενου «αδερφού») και επιτρέπει αυθαίρετες τροποποιήσεις, μια υλοποίηση πρέπει τουλάχιστον να αποθηκεύει στην μνήμη (**buffer**) το έγγραφο το οποίο έχει «διαβαστεί» μέχρι τότε, ή έστω κάποια αναλυμένη (**parsed**) μορφή του.

2.5.4.1 Μηχανές Διάταξης (Layout Engines)

Τα προγράμματα περιήγησης Ιστού βασίζονται στις «μηχανές διάταξης» για να αναλύσουν (**parse**) την **HTML** σε ένα μοντέλο **DOM**. Μερικές «μηχανές διάταξης» όπως η **Gecko**, η **Trident**/**MSHTML** και η **Presto** είναι συνδεδεμένες κυρίως ή και αποκλειστικά με ένα συγκεκριμένο πρόγραμμα περιήγησης όπως ο **Firefox**, ο **Internet Explorer** και ο **Opera** αντίστοιχα. Άλλες, όπως η **WebKit**, χρησιμοποιούνται από κοινού από έναν αριθμό περιηγητών, όπως ο **Safari** και ο **Chrome** της **Google**. Οι διάφορες «μηχανές διάταξης» υλοποιούν τα πρότυπα του μοντέλου **DOM** με ποικίλους βαθμούς «συμμόρφωσης».

2.5.4.2 Βιβλιοθήκες

Υλοποιήσεις του μοντέλου **DOM**:

- **Libxml2**
- **MSXML**
- **Xerces** (μια υλοποίηση του μοντέλου **DOM** σε γλώσσα **C++** με «δεσμεύσεις» (**bindings**) των γλωσσών **Java** και **Pearl**)

Διεπαφές Ανάπτυξης Εφαρμογών (**API**s) που εκθέτουν υλοποιήσεις του μοντέλου **DOM**:

- **JAXP** (**Java API** για Διεργασία (**Processing**) κώδικα **XML**) είναι μια **API** για την πρόσβαση παρόχων του μοντέλου **DOM**

Εναλλακτικές βιβλιοθήκες **XML** που βασίζονται σε δεντρική δομή και δεν υποστηρίζουν το μοντέλο **DOM**:

- **VTD-XML** (είναι μια βιβλιοθήκη Java η οποία προσφέρει εναλλακτική προβολή των εγγράφων **XML**, βασισμένη σε δέντρο)

2.6 Το Αντικείμενο XMLHttpRequest

2.6.1 Τι είναι το XMLHttpRequest;

Το **XMLHttpRequest** είναι μια Διεπαφή Ανάπτυξης Εφαρμογών (**Application Programming Interface** ή **API**) του μοντέλου **DOM** (**Document Object Model**). Μπορεί να χρησιμοποιηθεί μέσα σε μια γλώσσα προγραμματισμού ενός Περιηγητή Διαδικτύου, όπως η **JavaScript**, για να στείλει ένα αίτημα **HTTP** (**Hypertext Transfer Protocol**) ή **HTTPS** (**HTTP Secure**) απευθείας σε έναν Διακομιστή διαδικτύου και να φορτώσει τα δεδομένα απάντησής του Διακομιστή απευθείας πίσω στην γλώσσα. Από την στιγμή που τα δεδομένα βρεθούν μέσα στην γλώσσα, είναι διαθέσιμα τόσο ως ένα **XML** (**eXtensible Markup Language**) έγγραφο, αν η απάντηση είχε έγκυρη σήμανση **XML**, και ως «**απλό κείμενο**» (**plain text**). Τα δεδομένα **XML** μπορούν να χρησιμοποιηθούν για να προσαρμοστεί (να επεξεργαστεί, να παραποιηθεί) το τρέχον ενεργό έγγραφο στο παράθυρο του Περιηγητή, χωρίς να απαιτείται από τον Πελάτη να φορτώσει μια καινούργια έκδοση του εγγράφου της ιστοσελίδας. Επιπρόσθετα, τα δεδομένα σε «απλό κείμενο» μπορούν να αξιολογηθούν εντός της γλώσσας προγραμματισμού και να προσαρμόσουν και αυτά το έγγραφο. Για παράδειγμα, εάν χρησιμοποιείται η γλώσσα **JavaScript**, το «απλό κείμενο» μπορεί να διαμορφωθεί ως **JSON** (**JavaScript Object Notation**) από τον Διακομιστή διαδικτύου και να αξιολογηθεί από την **JavaScript** για να δημιουργηθεί ένα αντικείμενο δεδομένων (**data object**) για χρήση πάνω στο τρέχον μοντέλο **DOM**.

> *Ερ.: Το **AJAX** είναι απλά ένα άλλο όνομα για το **XMLHttpRequest**;*
>
> *Απ.: Όχι. Το **XMLHttpRequest** είναι μόνο ένα μέρος του **AJAX**. Το **XMLHttpRequest** είναι το τεχνικό εξάρτημα (**component**) που κάνει την ασύγχρονη επικοινωνία με την Διακομιστή δυνατή.*
>
> ***-- Jesse James Garrett***

2.6.2 Η Σχέση του XMLHttpRequest με το AJAX

Το αντικείμενο **XMLHttpRequest** παίζει ίσως τον πιο σημαντικό ρόλο στην διαδικτυακή τεχνική ανάπτυξης **AJAX**. Για να ληφθεί ή να σταλεί πληροφορία από/προς μια βάση δεδομένων ή ένα αρχείο στον Διακομιστή χρησιμοποιώντας παραδοσιακή **JavaScript**, έπρεπε να γίνουν τα παρακάτω βήματα:

- Αρχικά θα έπρεπε να δημιουργηθεί μια **HTML** φόρμα
- Ο χρήστης θα έπρεπε να πατήσει το κουμπί «**Υποβολή**» (**Submit button**) για να στείλει/λάβει την πληροφορία

- Στην συνέχεια να περιμένει τον Διακομιστή να απαντήσει
- Και τελικά θα φορτωνόταν μια νέα σελίδα με τα αποτελέσματα.

Λόγω του ότι ο Διακομιστής επέστρεφε μια νέα σελίδα κάθε φορά που ο χρήστης υπέβαλε μια νέα είσοδο (**input**), οι παραδοσιακές διαδικτυακές εφαρμογές έπρεπε να εκτελούνται αργά και έτσι έτειναν να γίνονται λιγότερο φιλικές προς τον χρήστη (**user-friendly**). Με το **AJAX**, η **JavaScript** μπορούσε να επικοινωνήσει απευθείας με τον Διακομιστή, και αυτό δεν θα μπορούσε να γίνει, παρά μόνο μέσω της χρήσης του αντικειμένου **XMLHttpRequest**. Γι αυτό το λόγο, το **AJAX** δεν θα υπήρχε εάν δεν είχε αναπτυχθεί το αντικείμενο **XMLHttpRequest**, καθώς όλη η φιλοσοφία του στηρίζεται στην ασύγχρονη επικοινωνία που αυτό του προσφέρει.

Με την χρήση του αντικειμένου **XMLHttpRequest**, μια ιστοσελίδα μπορεί να κάνει μια αίτηση προς, και να πάρει μια απάντηση από, έναν Διακομιστή διαδικτύου, χωρίς να ξαναφορτωθεί η σελίδα. Ο χρήστης θα μείνει στην ίδια σελίδα, και ο κώδικάς που τρέχει στο παρασκήνιο και ανταλλάσει σελίδες με τον Διακομιστή, θα λειτουργεί περνώντας απαρατήρητος γι αυτόν.

2.6.3 Ιστορία και Υποστήριξη

Η γενική ιδέα πίσω από το αντικείμενο **XMLHttpRequest** δημιουργήθηκε αρχικά από τους υπεύθυνους για την ανάπτυξη της υπηρεσίας ηλεκτρονικού ταχυδρομείου «**Outlook Web Access**», μέσα στα πλαίσια ανάπτυξης της τεχνολογίας Διακομιστή «**Microsoft Exchange Server 2000**» της γνωστής εταιρίας. Αναπτύχθηκε μια διεπαφή με το όνομα «**IXMLHTTPRequest**», η οποία στην συνέχεια ενσωματώθηκε στην δεύτερη έκδοση της βιβλιοθήκης **MSXML**, βασισμένη στην αρχική γενική ιδέα. Η δεύτερη έκδοση της βιβλιοθήκης **MSXML** κυκλοφόρησε με τον Περιηγητή «**Internet Explorer 5.0**» της εταιρίας, τον Μάρτιο του 1999, η οποία και επέτρεπε την πρόσβαση, μέσω του συστήματος (**framework**) **ActiveX**, στην διεπαφή **IXMLHTTPRequest** χρησιμοποιώντας το «**περικάλυμμα**» (**wrapper**) **XMLHTTP** της βιβλιοθήκης.

Η μη κερδοσκοπική οργάνωση **Mozilla Foundation** ανέπτυξε και ενσωμάτωσε μια διεπαφή με το όνομα **nslXMLHttpRequest** μέσα στην «**μηχανή διάταξης**» (**layout engine**) **Gecko**. Αυτή η διεπαφή είχε μοντελοποιηθεί έτσι ώστε η λειτουργία της να πλησιάζει όσο το δυνατόν περισσότερο σε αυτήν της διεπαφής **IXMLHTTPRequest** της **Microsoft**. Η **Mozilla** δημιούργησε ένα «**περικάλυμμα**» (**wrapper**) για την χρήση αυτής της διεπαφής μέσω ενός αντικειμένου της **JavaScript**, το οποίο ονόμασε **XMLHttpRequest**. Το αντικείμενο **XMLHttpRequest** έγινε προσιτό με την κυκλοφορία της έκδοσης **0.6** της μηχανής **Gecko**, στις 6 Δεκεμβρίου του 2000, δεν ήταν όμως πλήρως λειτουργικό. Για να γίνει, χρειάστηκε περίπου ενάμιση χρόνος, έως ότου η **Mozilla** να κυκλοφόρησε την έκδοση **1.0** της μηχανής

Gecko, τον Ιούνιο του 2002. Έπειτα, το αντικείμενο **XMLHttpRequest** έγινε ένα «de facto» πρότυπο μεταξύ των υπόλοιπων κυριοτέρων «**αντιπρόσωπων χρήστη**» (**user agents**). Για το λειτουργικό σύστημα της εταιρίας **Apple**, το **Mac OS**, το πρότυπο ενσωματώθηκε στο περιηγητή διαδικτύου **Safari** από την έκδοσή **1.2** που κυκλοφόρησε τον Φεβρουάριο του 2004, και στον περιηγητή διαδικτύου **iCab** από την έκδοση **3.0b352** που κυκλοφόρησε τον Σεπτέμβριο του 2005. Το πρότυπο ενσωματώθηκε και στον περιηγητή ιστού **Opera**, από την έκδοση **8.0** που κυκλοφόρησε τον Απρίλιο του 2005, ενώ δεν άργησε να ενσωματωθεί και στον περιηγητή ιστού **Konqueror** του λειτουργικού συστήματος **Linux**.

Η κοινοπραξία **World Wide Web Consortium** (**W3C**) δημοσίευσε ένα «σε εξέλιξη» Προσχέδιο (**Working Draft**) προδιαγραφής για το αντικείμενο **XMLHttpRequest** στις 5 Απρίλιο του 2006, το οποίο είχε συνταχθεί από την **Anne Van Kesteren** της εταιρίας **Opera Software** και τον **Dean Jackson** της **W3C**. Σκοπός του προσχεδίου, σύμφωνα με τους συντάκτες του ήταν «η τεκμηρίωση ενός ελάχιστου συνόλου από συμβατά χαρακτηριστικά (**interoperable features**) που θα βασίζονταν πάνω στις υπάρχοντες υλοποιήσεις, το οποίο θα επέτρεπε στους προγραμματιστές διαδικτύου να χρησιμοποιούν αυτά τα χαρακτηριστικά χωρίς την χρήση κώδικα ο οποίος θα ήταν συγκεκριμένης πλατφόρμας (**platform-specific code**). Η τελευταία αναθεώρηση της προδιαγραφής του αντικειμένου **XMLHttpRequest** έγινε στις 19 Νοεμβρίου του 2009, το οποίο κατατάσσεται ακόμα στο στάδιο εξέλιξης.

Παράλληλα, η κοινοπραξία **W3C** έχει δημοσιεύσει και ένα επιπλέον προσχέδιο προδιαγραφής για το αντικείμενο **XMLHttpRequest**, στις 25 Φεβρουαρίου του 2008, το οποίο επίσης βρίσκεται σε στάδιο εξέλιξης και είχε τίτλο «**XMLHttpRequest Επίπεδο 2**». Το «**Επίπεδο 2**» αποτελείται από επεκτάσεις στην λειτουργικότητα του αντικειμένου **XMLHttpRequest**. Μερικές από αυτές τις επεκτάσεις είναι τα συμβάντα προόδου (**progress events**), η υποστήριξη για αιτήματα μεταξύ δικτυακών τόπων και η μεταχείριση των ροών των **bytes** (**byte streams**). Η τελευταία αναθεώρηση στην προδιαγραφή του «**Επίπεδου 2**» έγινε στις 30 Σεπτεμβρίου του 2008, ενώ εξακολουθεί να χαρακτηρίζεται ως προσχέδιο «εν εξελίξει».

Η **Microsoft** πρόσθεσε το αναγνωριστικό (**identifier**) του αντικειμένου **XMLHttpRequest** στις δικές της γλώσσες προγραμματισμού με τον περιηγητή **Internet Explorer 7.0**, ο οποίος βγήκε στην κυκλοφορία τον Οκτώβριο του 2006, έξι μήνες μετά το αρχικό προσχέδιο της προδιαγραφής του **XMLHttpRequest**. Εδώ πρέπει να επισημανθεί το πρόβλημα που υπάρχει με τις προγενέστερες εκδόσεις του περιηγητή **Internet Explorer**, **5.0**, **5.5** και **6**. Στις εκδόσεις **5** και **6** δεν καθορίζεται το αναγνωριστικό για το αντικείμενο **XMLHttpRequest** μέσα στις γλώσσες προγραμματισμού τους, καθώς το αναγνωριστικό δεν είχε γίνει πρότυπο τον καιρό που αυτές οι εκδόσεις κυκλοφόρησαν. Για να επιτευχθεί η συμβατότητα «προς τα πίσω» (**backward compatibility**) θα πρέπει να γίνει μια ανίχνευσή αντικειμένου (**object detection**)

για την διαπίστωση εάν το αναγνωριστικό του **XMLHttpRequest** υφίσταται ή όχι. Οι ιστοσελίδες που κάνουν χρήση του **XMLHttpRequest** ή του **XMLHTTP** μπορούν να μετριάσουν τις τρέχουσες μικροδιαφορές στις υλοποιήσεις, είτε με την ενθυλάκωση του αντικειμένου **XMLHttpRequest** στο εσωτερικό ενός «**περικαλύμματος**» (**wrapper**) της **JavaScript**, είτε με την χρήση ενός υπάρχοντος πλαισίου (**framework**) που την κάνει. Και στις δύο περιπτώσεις, το «**περικάλυμμα**» πρέπει να αναγνωρίζει τις ικανότητες που έχουν οι τρέχουσες υλοποιήσεις και να λειτουργεί μέσα στα πλαίσια των αναγκών του.

2.6.4 Το Αίτημα HTTP

Σε αυτήν την υποενότητα θα περιγραφεί ο τρόπος με τον οποίο λειτουργεί ένα αίτημα που κάνει χρήση του αντικειμένου **XMLHttpRequest** στα πλαίσια ενός «**αντιπρόσωπου χρήστη**» (**user agent**) που «συμμορφώνεται» με το «εν εξελίξει» προσχέδιο της κοινοπραξία **W3C**. Καθώς το πρότυπο της **W3C** για το αντικείμενο **XMLHttpRequest** είναι ακόμα ένα προσχέδιο, οι «**αντιπρόσωποι χρήστη**» δεν είναι υποχρεωμένοι να «υιοθετήσουν» όλες τις λειτουργίες που ορίζει η **W3C**, επομένως οποιοδήποτε από τα παρακάτω υπόκεινται σε αλλαγές. Ιδιαίτερη προσοχή πρέπει να δίνεται κατά τον προγραμματισμό με το αντικείμενο **XMLHttpRequest** που αφορά πολλαπλούς «**αντιπροσώπους χρήστη**». Παρακάτω θα επιχειρηθεί η καταγραφή των αντιφάσεων μεταξύ των κυριότερων «**αντιπρόσωπων χρήστη**».

2.6.4.1 Η Μέθοδος «open»

Τα αιτήματα **HTTP** και **HTTPS** του αντικειμένου **XMLHttpRequest** πρέπει να αρχικοποιηθούν μέσω της μεθόδου «**open**». Αυτή η μέθοδος πρέπει να κληθεί πριν από την πραγματική αποστολή μιας αίτησης, για να επικυρώσει και να αναλύσει την μέθοδο αίτησης, την διεύθυνση **URL**, και την διεύθυνση **URI** (**Uniform Resource Identifier**, **Ομοιόμορφο Αναγνωριστικό Πηγής**) των στοιχείων του χρήστη, τα οποία και πρόκειται να χρησιμοποιηθούν για την αίτηση. Αυτή η μέθοδος δεν διαβεβαιώνει ότι η διεύθυνση **URL** υπάρχει ή ότι τα στοιχεία του χρήστη είναι σωστά. Η μέθοδος μπορεί να δεχτεί έως και πέντε παραμέτρους, απαιτεί όμως μόνο δύο για να αρχικοποιήσει μια αίτηση.

Η πρώτη παράμετρος της μεθόδου είναι μια συμβολοσειρά κειμένου (**text string**) που υποδεικνύει την μέθοδο αίτησης **HTTP** που χρησιμοποιεί. Οι μέθοδοι αίτησης που πρέπει να υποστηρίζονται από έναν «συμμορφωμένο» «**αντιπρόσωπο χρήστη**», όπως περιγράφονται από το προσχέδιο της **W3C** για το αντικείμενο **XMLHttpRequest**, έχουν καταγραφεί επί του παρόντος ως οι εξής:

- **GET** (Υποστηρίζεται από τον **IE7+**, **Mozilla 1+**)
- **POST** (Υποστηρίζεται από τον **IE7+**, **Mozilla 1+**)

- **HEAD** (Υποστηρίζεται από τον **IE7+**)
- **PUT**
- **DELETE**
- **OPTIONS** (Υποστηρίζεται από τον **IE7+**)

Ωστόσο, οι μέθοδοι αίτησης δεν περιορίζονται σε εκείνες που αναγράφονται παραπάνω. Το προσχέδιο της **W3C** δηλώνει ότι ένα πρόγραμμα περιήγησης μπορεί να υποστηρίζει επιπρόσθετες μεθόδους αίτησης, κατά την δική τους κρίση.

Η δεύτερη παράμετρος της μεθόδου είναι άλλη μία συμβολοσειρά κειμένου, η οποία υποδεικνύει την διεύθυνση **URL** της αίτησης **HTTP.** Η **W3C** προτείνει ότι τα προγράμματα περιήγησης θα πρέπει να εμφανίζουν ένα σφάλμα (**raise an error**) και να μην επιτρέπουν την αίτηση μια διεύθυνσης **URL** η οποία θα αναφέρεται είτε σε διαφορετική θύρα (**port**), είτε σε διαφορετικό συστατικό «**ihost**» (**ihost component**) της διεύθυνσης **URI** από το τρέχον έγγραφο.

Η τρίτη παράμετρος είναι μια «λογική» («**Boolean**») τιμή η οποία υποδεικνύει εάν η αίτηση είναι, ή δεν είναι, ασύγχρονη. Αυτή η παράμετρος δεν είναι απαιτούμενη σύμφωνα με το προσχέδιο της **W3C**. Η προεπιλεγμένη τιμή της παραμέτρου θα πρέπει να θεωρείται ότι είναι η «**αληθής**» («**true»**) από έναν «**αντιπρόσωπο χρήστη**» που είναι συμμορφωμένος με την **W3C**, όταν η παράμετρος δεν έχει οριστεί. Μια ασύγχρονη αίτηση («**true**») δεν θα περιμένει για μια απάντηση του Διακομιστή πριν να συνεχίσει με την εκτέλεση του τρέχοντος κομματιού κώδικα. Αντιθέτως, θα επικαλεστεί τον «**ακροατή συμβάντων**» (**event listener**) «**onreadystatechange**» του αντικειμένου **XMLHttpRequest** κατά την διάρκεια των διάφορων σταδίων της αίτησης. Μια σύγχρονη αίτηση («**false**») ωστόσο, θα σταματήσει την εκτέλεση του τρέχοντος κομματιού κώδικα έως ότου η αίτηση έχει ολοκληρωθεί, χωρίς να επικαλεστεί τον «**ακροατή συμβάντων**» «**onreadystatechange**».

Η τέταρτη και πέμπτη παράμετρος αφορούν τον χρήστη και τον κωδικό πρόσβασης, αντίστοιχα, της διεύθυνσης **URI**. Αυτές οι παράμετροι δεν απαιτούνται και πρέπει να χρησιμοποιείται ως προεπιλογή ο τρέχον χρήστης και κωδικός πρόσβασης του εγγράφου, εφόσον αυτά δεν παρέχονται, όπως ορίζεται από το προσχέδιο της **W3C.**

2.6.4.2 Η Μέθοδος «setRequestHeader»

Με την επιτυχή αρχικοποίηση μιας αίτησης, η μέθοδος «**setRequestHeader**» του αντικειμένου **XMLHttpRequest** μπορεί να κληθεί για να στείλει «**κεφαλίδες**» (**headers**) του πρωτοκόλλου **HTTP** με την αίτηση. Η πρώτη παράμετρος αυτής της μεθόδου είναι το όνομα τις συμβολοσειράς κειμένου της «**κεφαλίδας**». Η δεύτερη παράμετρος είναι η τιμή της συμβολοσειράς κειμένου. Η μέθοδος αυτή πρέπει να κληθεί για κάθε «**κεφαλίδα**» που χρειάζεται να σταλεί με την αίτηση. Οποιεσδήποτε «**κεφαλίδες**» που συνάπτονται εδώ θα

αφαιρεθούν την επόμενη φορά που θα κληθεί η μέθοδος «**open**» στην περίπτωση ενός συμμορφωμένου με την **W3C** «**αντιπρόσωπου χρήστη**».

2.6.4.3 Η Μέθοδος «send»

Για να σταλεί μια αίτηση **HTTP**, πρέπει να κληθεί η μέθοδος «**send**» του αντικειμένου **XMLHttpRequest**. Αυτή η μέθοδος δέχεται μία μόνο παράμετρο η οποία περιέχει το περιεχόμενο που πρέπει να σταλθεί με την αίτηση. Η παράμετρος αυτή μπορεί να παραληφθεί εάν δεν υπάρχει περιεχόμενο που πρέπει να σταλθεί. Το προσχέδιο της **W3C** αναφέρει ότι η παράμετρος αυτή μπορεί να είναι οποιουδήποτε τύπου που είναι διαθέσιμος στην γλώσσα προγραμματισμού, αρκεί να μπορεί να μετατραπεί σε συμβολοσειρά κειμένου, με εξαίρεση το αντικείμενο εγγράφου **DOM** (**DOM document**). Εάν ένας «**αντιπρόσωπος χρήστη**» δεν μπορέσει να μετατρέψει αυτήν την παράμετρο σε συμβολοσειρά, τότε η παράμετρος πρέπει να αγνοηθεί.

Εάν η παράμετρος είναι ένα αντικείμενο εγγράφου **DOM**, ο «**αντιπρόσωπος χρήστη**» πρέπει να διασφαλίσει ότι το έγγραφο έχει μετατραπεί σε «ορθώς-μορφοποιημένο» (**well-formed**) κώδικα **XML**, χρησιμοποιώντας την κωδικοποίηση που υποδεικνύεται από την ιδιότητα (**property**) «**inputEncoding**» του αντικειμένου του εγγράφου. Εάν η κεφαλίδα αίτησης «**Content-Type**» δεν έχει προστεθεί ακόμα μέσω της μεθόδου «**setRequestHeader**», θα πρέπει να προστεθεί αυτόματα από έναν συμμορφωμένο «**αντιπρόσωπο χρήστη**» ως «**application/xml;charset= charset**», όπου «**charset**» είναι η κωδικοποίηση που χρησιμοποιήθηκε για να κωδικοποιηθεί το έγγραφο.

2.6.4.4 Ο Ακροατής Συμβάντων «onreadystatechange»

Εάν η μέθοδος «**open**» του αντικειμένου **XMLHttpRequest** κληθεί με την τρίτη παράμετρο να έχει οριστεί ως «**αληθής**» για μια ασύγχρονη αίτηση, τότε ο «**ακροατής συμβάντων**» «**onreadystatechange**» θα κληθεί αυτομάτως για κάθε μία από τις ακόλουθες ενέργειες οι οποίες αλλάζουν την ιδιότητα (**property**) «**readyState**» του αντικειμένου **XMLHttpRequest**.

- Μετά από την επιτυχή κλήση της μεθόδου «**open**», η ιδιότητα «**readyState**» του αντικειμένου **XMLHttpRequest** πρέπει να οριστεί με την τιμή «**1**».
- Μετά από την κλήση της μεθόδου «send» και την λήψη των «κεφαλίδων» **HTTP** της απάντησης, η ιδιότητα «**readyState**» του αντικειμένου **XMLHttpRequest** πρέπει να οριστεί με την τιμή «**2**».
- Μόλις το περιεχόμενο της **HTTP** απάντησης αρχίζει να φορτώνεται, η ιδιότητα «**readyState**» του αντικειμένου **XMLHttpRequest** πρέπει να οριστεί με την τιμή «**3**».

- Μόλις ολοκληρωθεί η φόρτωση του περιεχόμενου της **HTTP** απάντησης, η ιδιότητα «**readyState**» του αντικειμένου **XMLHttpRequest** πρέπει να οριστεί με την τιμή «**4**».

Οι κυριότεροι «**αντιπρόσωποι χρήστη**» είναι ασυνεπείς με τον χειρισμό του «**ακροατή συμβάντων**» «**onreadystatechange**».

2.6.4.5 Η Απάντηση HTTP

Μετά από μία επιτυχημένη και ολοκληρωμένη κλήση της μεθόδου «**send**» του αντικειμένου **XMLHttpRequest**, εάν η απάντηση του Διακομιστή ήταν έγκυρος κώδικας **XML** και η «**κεφαλίδα**» «**Content-Type**» που έχει σταλθεί από τον Διακομιστή έχει γίνει κατανοητή από τον «**αντιπρόσωπο χρήστη**» ως ένας **XML** «**τύπος μέσων Διαδικτύου**» (**XML Internet media type**), η ιδιότητα (**property**) «**responseXML**» του αντικειμένου **XMLHttpRequest** θα περιέχει ένα αντικείμενο εγγράφου **DOM** (**DOM document object**). Μια άλλη ιδιότητα, η «**responseText**», θα περιέχει την απάντηση του Διακομιστή σε «απλό κείμενο», από έναν συμμορφωμένο «**αντιπρόσωπο χρήστη**», ανεξάρτητα από το εάν ήταν ή δεν ήταν κατανοητή ως κώδικας **XML**.

2.7 Η βιβλιοθήκη jQuery

2.7.1 Τι είναι η jQuery;

Η **jQuery** είναι μια «ελαφριά» βιβλιοθήκη της γλώσσας **JavaScript**, η οποία είναι ανεξάρτητη προγράμματος περιήγησης και δίνει έμφαση στην αλληλεπίδραση μεταξύ της **JavaScript** και της **HTML**. Κυκλοφόρησε από τον **John Resig** τον Ιανουάριο του 2006 στην συνδιάσκεψη **BarCamp** που έλαβε χώρα στη Νέα Υόρκη. Η **jQuery** είναι η πιο δημοφιλής βιβλιοθήκη της **JavaScript** σήμερα, καθώς χρησιμοποιείται από το 27% των δέκα χιλιάδων πιο δημοφιλών ιστοσελίδων παγκοσμίως.

Η **jQuery** διανέμεται δωρεάν και ανήκει στην κατηγορία του λογισμικού ανοικτού κώδικα (**open source software**). Κυκλοφορεί με δύο άδειες χρήσης, την «**MIT License**» και την «**GNU General Public License, Version 2**». Το συντακτικό της **jQuery** έχει σχεδιαστεί να κάνει ευκολότερη την πλοήγηση σε ένα έγγραφο, την επιλογή στοιχείων του μοντέλου **DOM**, την δημιουργία εφέ κινήσεως (**animations**), της διαχείρισης των συμβάντων, και της ανάπτυξης εφαρμογών **AJAX**. Επίσης, η **jQuery** παρέχει δυνατότητες στους προγραμματιστές να δημιουργήσουν «συμπληρωματικά προγράμματα» (**plug-ins**) πάνω από την βιβλιοθήκη της **JavaScript**. Εκμεταλλευόμενοι αυτήν την δυνατότητα, οι προγραμματιστές μπορούν να δημιουργήσουν «αφηρημένες ιδέες» («**abstractions**») για

χαμηλού επιπέδου αλληλεπίδραση και εφέ κίνησης, καθώς και προηγμένα εφέ και βοηθητικές εφαρμογές (**widgets**) υψηλού επιπέδου. Με αυτό τον τρόπο συμβάλει στην δημιουργία ισχυρών και δυναμικών ιστοσελίδων.

Οι εταιρίες **Microsoft** και **Nokia** έχουν ανακοινώσει ότι σχεδιάζουν να ενσωματώσουν την **jQuery** μέσα στις πλατφόρμες τους. Η **Microsoft** σκοπεύει να την ενσωματώσει αρχικά στο «**Visual Studio**» για χρήση με το «πλαίσιο» (**framework**) **AJAX** και το «πλαίσιο» **MVC** της **ASP.NET**. Η **Nokia** θα το ενσωματώσει στην πλατφόρμα της «**Web Run-Time**». Το δωρεάν και ανοικτού κώδικα «πλαίσιο» **Seaside**, παρέχει πλήρη ενσωμάτωση της **jQuery** επιτρέποντας την ανάπτυξη διαδικτυακών εφαρμογών κάνοντας χρήση εξ ολοκλήρου της γλώσσας **Smalltalk**.

2.7.2 Χαρακτηριστικά

Η βιβλιοθήκη **jQuery** περιέχει τα ακόλουθα χαρακτηριστικά:

- Επιλογή στοιχείων του μοντέλου **DOM** κάνοντας χρήση της ανοικτού τύπου «μηχανής επιλογέα» (**selector engine**) **Sizzle**, η οποία είναι ανεξάρτητη προγράμματος περιήγησης και αποτελεί ένα υποπροϊόν (**spin-off**) της **jQuery**
- Διάσχιση (**traversal**) και τροποποίηση του μοντέλου **DOM** (συμπεριλαμβανομένης και της υποστήριξης για φύλλα **CSS** των εκδόσεων 1-3)
- Συμβάντα
- Χειρισμό των φύλλων στυλ **CSS**
- Απλά εφέ και εφέ κίνησης
- **AJAX**
- Επεκτασιμότητα μέσω των «συμπληρωματικών προγραμμάτων» (**plug-ins**)
- Βοηθητικά Εργαλεία (**Utilities**) όπως ο εντοπισμός της έκδοσης του προγράμματος περιήγησης και η συνάρτηση «**each**».

2.7.3 Χρήση

Η βιβλιοθήκη **jQuery** υπάρχει συνήθως ως ένα ενιαίο αρχείο της **JavaScript**, το οποίο περιέχει όλα τα κοινά μοντέλα **DOM**, τα Συμβάντα, τα Εφέ, και τις συναρτήσεις του **AJAX**. Μπορεί να συμπεριληφθεί σε μία ιστοσελίδα απλά με την χρήση της παρακάτω γραμμής κώδικα σήμανσης:

```
<script type="text/javascript" src="jQuery.js"></script>
```

Η **JQuery** μπορεί επίσης να προσπελαθεί, να φορτωθεί και να εκτελεστεί με τον ίδιο ακριβώς τρόπο που μπορεί και η γλώσσα **JavaScript**.

Επιπροσθέτως, η **jQuery** μπορεί να φορτωθεί χρησιμοποιώντας τις βιβλιοθήκες της **Google** για το **AJAX**, τις **Google AJAX Libraries API**, με την χρήση του ακόλουθου κώδικα σήμανσης:

```
<script type="text/javascript"
src="http://www.google.com/jsapi"></script>
<script>
google.load("jquery", "1.3.2");
</script>
```

Η **jQuery** έχει δύο στυλ αλληλεπίδρασης:

- Μέσω της συνάρτησης «**$**», η οποία είναι μια «μέθοδος εργοστάσιο» (**factory method**) για το αντικείμενο της **jQuery**. Αυτές οι συναρτήσεις, που συχνά αποκαλούνται «εντολές», μπορούν να χρησιμοποιηθούν αλυσιδωτά (**chainable**). Κάθε μια από αυτές επιστρέφουν το αντικείμενο της **jQuery** (**jQuery object**).
- Μέσω των συναρτήσεων που έχουν ως πρόθεμα το «**$**.» («**$. –prefixed functions**»). Αυτές είναι «συναρτήσεις εργαλεία» (**utility functions**) οι οποίες δεν δουλεύουν με το αντικείμενο **jQuery** αυτό καθ' εαυτό.

Μια τυπική ροή εργασίας για τον χειρισμό πολλαπλών κόμβων του μοντέλου **DOM**, ξεκινάει με την κλήση της συνάρτησης «**$**» με μια «συμβολοσειρά επιλογέα» σε γλώσσα **CSS** (**CSS selector string**), που έχει ως αποτέλεσμα την παραπομπή σε κανένα ή περισσότερα στοιχεία της σελίδας **HTML** από το αντικείμενο της **jQuery**. Αυτό το σύνολο κόμβων μπορεί να διαχειριστεί με την εφαρμογή μεθόδων «περίστασης» (**instance methods**) στο αντικείμενο της **jQuery**, ή μπορεί να διαχειριστούν οι ίδιοι οι κόμβοι. Για παράδειγμα ο παρακάτω κώδικας…:

```
$("div.test").add("p.quote").addClass("blue").slideDown("slow");
```

…βρίσκει την ένωση όλων των ετικετών (**tags**) «**div**» με την ιδιότητα κλάσης (**class attribute**) «**test**» και όλων των ετικετών «**p**» με την ιδιότητα κλάσης «**quote**» του φύλλου **CSS**, προσθέτει την ιδιότητα κλάσης «**blue**» σε κάθε στοιχείο που έχει βρεθεί, και στη συνέχεια τα «κυλάει» προς τα κάτω με ένα εφέ κίνησης. Οι συναρτήσεις «**$**» και «**add**» επηρεάζουν το σύνολο που έχει βρεθεί, ενώ οι συναρτήσεις «**addClass**» και «**slideDown**» επηρεάζουν τους κόμβους που αναφέρονται.

Οι μέθοδοι με το πρόθεμα «**$.**» είναι μέθοδοι ευκολίας ή επηρεάζουν δημόσιες ιδιότητες (**global properties**) και συμπεριφορές. Για παράδειγμα, ο κώδικας που ακολουθεί παρουσιάζει μια συνάρτηση χαρτογράφησης της **jQuery** που ονομάζεται «**each**» και «γράφει» τους αριθμούς **234** στο τρέχον έγγραφο:

```
$.each([1,2,3], function() {
  document.write(this + 1);
});
```

Με την **jQuery** γίνεται πολύ απλή η εκτέλεση ερωτημάτων (**queries**) **AJAX** τα οποία εκτελούνται ανεξάρτητα με το πρόγραμμα περιήγησης που χρησιμοποιείται. Αυτό γίνεται με την χρήσης της «**$.ajax**» και συναφών μεθόδων για να φορτωθούν και να χειριστούν απομακρυσμένα δεδομένα. Το ακόλουθο κομμάτι κώδικα θα δημιουργήσει μια αίτηση για την διεύθυνση «**some.php**» η οποία θα σταλεί στον Διακομιστή με τις παραμέτρους «**name=John**» και «**location=Boston**» και όταν η αίτηση ολοκληρωθεί με επιτυχία, η συνάρτηση «**success**» θα εκτελεστεί για να ειδοποιήσει τον χρήστη:

```
$.ajax({
  type: "POST",
  url: "some.php",
  data: "name=John&location=Boston",
  success: function(msg){
    alert( "Data Saved: " + msg );
  }
});
```

Τελειώνοντας, αξίζει να αναφερθεί πως είναι εντυπωσιακό το γεγονός ότι η βιβλιοθήκη **jQuery** συγκεντρώνει τόσες πολλές λειτουργίες και ευκολίες, ενώ παράλληλα καταφέρνει να τις «μαζεύει» σε ένα και μοναδικό αρχείο πυρήνα του οποίου το μέγεθος δεν ξεπερνάει τα 25 kilobytes.

2.8 Η γλώσσα σήμανσης XML

Η γλώσσα **XML** (**eXtensible Markup Language**) είναι ουσιαστικά ένα σύνολο από κανόνες για την ηλεκτρονική κωδικοποίηση εγγράφων. Ορίζεται μέσα από την προδιαγραφή «**XML 1.0**», η οποία έχει συνταχθεί από την κοινοπραξία **W3C**, και από πολλές άλλες σχετικές προδιαγραφές. Όλες οι προδιαγραφές που την ορίζουν εντάσσονται στα δωρεάν, ανοικτά πρότυπα (**open standards**).

Η **XML** σχεδιάστηκε με σκοπό να δοθεί έμφαση στην απλότητα, στην γενικότητα, και στην ευχρηστία γύρω από το διαδίκτυο. Πρόκειται για μια μορφή δεδομένων σε κείμενο, με ισχυρή υποστήριξη για τις γλώσσες του κόσμου μέσω του προτύπου «**Unicode**». Παρόλο που η σχεδίαση της **XML** επικεντρώνεται στα έγγραφα, η γλώσσα χρησιμοποιείται ευρέως για την αναπαράσταση αυθαίρετων δομών δεδομένων, όπως για παράδειγμα στις διαδικτυακές υπηρεσίες (**web services**). Υπάρχει μια ποικιλία από προγραμματιστικές διεπαφές τις οποίες οι προγραμματιστές υλικού μπορούν να χρησιμοποιήσουν για να έχουν πρόσβαση σε

δεδομένα **XML**, καθώς και αρκετά «συστήματα σχήματος» (**schema systems**) που έχουν σχεδιαστεί να βοηθάνε στον καθορισμό των γλωσσών που βασίζονται στην **XML**.

Μέχρι και σήμερα, έχουν αναπτυχθεί εκατοντάδες γλώσσες οι οποίες είναι βασισμένες στην **XML**, συμπεριλαμβανομένων των **RSS**, **ATOM**, **SOAP** και **XHTML**. Πρότυπα που είναι βασισμένα στην **XML** έχουν γίνει η προεπιλογή για πολλά εργαλεία εφαρμογών γραφείου, όπως το **Microsoft Office** (**Office Open XML**), το **OpenOffice.org** (**OpenDocument**), και το **iWork** της **Apple**.

2.9 Το μορφότυπο JSON

Το μορφότυπο **JSON** (**JavaScript Object Notation**), είναι ένα «ελαφρύ» πρότυπο ανταλλαγής δεδομένων για τον υπολογιστή. Είναι ένα πρότυπο που βασίζεται σε κείμενο (**text-based**), είναι αναγνώσιμο από τον άνθρωπο (**human-readable**), και χρησιμοποιείται για την αναπαράσταση απλών δομών δεδομένων και προσεταιριστικών συστοιχιών (**associative arrays**) που ονομάζονται αντικείμενα (**objects**).

Το πρότυπο **JSON** ορίστηκε αρχικά από τον **Douglas Crockford** (πρότυπο **RFC 4627**). Ο επίσημος τύπος διαδικτυακού μέσου (**Internet media type**) για το **JSON** ορίζεται ως «**application/json**» ενώ η κατάληξη ενός αρχείου **JSON** έχει οριστεί η «**.json**». Το πρότυπο **JSON** χρησιμοποιείται συχνά για την σειριακή τοποθέτηση (**serialization**) και την μετάδοση δομημένων δεδομένων μέσω μιας σύνδεσης δικτύου. Χρησιμοποιείται κυρίως για την μετάδοση δεδομένων μεταξύ ενός Διακομιστή και μιας διαδικτυακής εφαρμογής, χρησιμεύοντας ως μια εναλλακτική επιλογή για την **XML**.

2.9.1 Ιστορία

Παρόλο που το **JSON** βασίστηκε σε ένα υποσύνολο της γλώσσας προγραμματισμού **JavaScript** (συγκεκριμένα στο πρότυπο «**Standard ECMA-262 Έκδοση 3η** – Δεκέμβριος 1999») και συνήθως χρησιμοποιείται σε συνδυασμό με αυτήν την γλώσσα, θεωρείτε ένα πρότυπο που είναι ανεξάρτητο γλώσσας. Ο κώδικας για την ανάλυση και παραγωγή δεδομένων σε **JSON** είναι άμεσα διαθέσιμος για μια μεγάλη ποικιλία γλωσσών προγραμματισμού. Η διεύθυνση http://json.org παρέχει ένα πλήρη κατάλογο από υπάρχοντες βιβλιοθήκες **JSON**, οι οποίες είναι οργανωμένες ανά γλώσσα. Τον Δεκέμβριο του 2005, η εταιρία «**Yahoo!**» άρχισε να προσφέρει ορισμένες από τις διαδικτυακές τις υπηρεσίες σε πρότυπο **JSON**. Τον Δεκέμβριο του 2006, η εταιρία **Google** άρχισε να προσφέρει «τροφοδοτήσεις» (**feeds**) του προτύπου **JSON** για το διαδικτυακό πρωτόκολλό της «**GData**».

2.9.2 Τύποι Δεδομένων και Συντακτικό

Οι βασικοί τύποι δεδομένων του **JSON** είναι οι εξής:

- Αριθμός (**Number**) (ακέραιος ή πραγματικός)(**integer or real**)
- Συμβολοσειρά (**String**) (περιβάλλεται από διπλά «**”**», είναι προτύπου **Unicode**, με «χαρακτήρα διαφυγής» το «\»)
- Λογικός (**Boolean**) (αληθής (**true**) ή ψευδής (**false**))
- Συστοιχία (**Array**) (μια ταξινομημένη σειρά από τιμές, οι οποίες διαχωρίζονται με κόμμα και περιβάλλονται από τους χαρακτήρες «**[**» και «**]**»)
- Αντικείμενο (**Object**) (μια συλλογή από ζευγάρια «κλειδιού:τιμής» (**key:value**) τα οποία διαχωρίζονται με κόμμα και περιβάλλονται από τους χαρακτήρες «**{**» και «**}**»)
- Κενό (**null**)

Το παράδειγμα που ακολουθεί δείχνει μια αναπαράσταση ενός αντικειμένου σε **JSON**, το οποίο περιγράφει ένα άτομο. Το αντικείμενο χρησιμοποιεί πεδία συμβολοσειρών (**string fields**) για το «όνομα» και το «επώνυμο», εμπεριέχει ένα άλλο αντικείμενο το οποίο αναπαριστά την διεύθυνση του ατόμου, και περιέχει και μια λίστα (**array**) από αντικείμενα με αριθμούς τηλεφώνου.

```
{
      "firstName": "John",
      "lastName": "Smith",
      "age": 25,
      "address": {
          "streetAddress": "21 2nd Street",
          "city": "New York",
          "state": "NY",
          "postalCode": "10021"
      },
      "phoneNumber": [
          { "type": "home", "number": "212 555-1234" },
          { "type": "fax", "number": "646 555-4567" }
      ]
 }
```

Ένα πιθανό ισοδύναμο παράδειγμα σε κώδικα **XML** και μπορούσε να είναι:

```
<Person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumber type="home">212 555-1234</phoneNumber>
  <phoneNumber type="fax">646 555-4567</phoneNumber>
</Person>
```

Σύμφωνα με το πρότυπο **RFC**, ο τύπος διαδικτυακού μέσου (**Internet media type ή MIME**) που πρέπει να χρησιμοποιείται όταν μεταδίδεται ένα αρχείο **JSON** μέσω του πρωτοκόλλου **HTTP** είναι το «**application/json**».

Δεδομένου ότι το **JSON** είναι ένα υποσύνολο της **JavaScript**, υπάρχει η δυνατότητα, αλλά δεν προτείνεται, να αναλυθεί το κείμενο **JSON** σε ένα αντικείμενο, με την κλήση της συνάρτησης «**eval()**» της **JavaScript**. Για παράδειγμα, ας υποτεθεί ότι το παραπάνω τμήμα κειμένου σε **JSON** περιλαμβάνεται σε μια μεταβλητή τύπου συμβολοσειράς (**string variable**) της **JavaScript** που ονομάζεται «**contact**». Η δημιουργία ενός αντικειμένου **JavaScript** από τα δεδομένα **JSON** με το όνομα «**p**», θα μπορούσε να γίνει με την παρακάτω δήλωση:

```
var p = eval("(" + contact + ")");
```

Η μεταβλητή «**contact**» πρέπει να περιβάλλεται από παρενθέσεις για να αποφευχθεί η ασάφεια στο συντακτικό της **JavaScript**.

Ωστόσο, ο προτεινόμενος τρόπος είναι η χρήση ενός αναλυτή (**parser**) **JSON**. Γενικά θα πρέπει να αποφεύγεται η χρήση τη συνάρτησης «**eval()**», εκτός εάν ο Πελάτης εμπιστεύεται απόλυτα την πηγή του κειμένου ή αν πρέπει να αναλύσει και να αποδεχτεί κείμενο το οποίο δεν είναι απολύτως συμβατό με το **JSON**. Ένας αναλυτής **JSON** δέχεται μόνο έγκυρα δεδομένα **JSON**, εμποδίζοντας έτσι την εκτέλεση ενδεχόμενου κακόβουλου κώδικα.

Οι σύγχρονοι περιηγητές, όπως ο **Firefox 3.5** και ο **Internet Explorer 8**, περιλαμβάνουν ειδικά χαρακτηριστικά για την ανάλυση δεδομένων **JSON**. Εφόσον η εγγενής υποστήριξη είναι πιο αποδοτική και ασφαλής από την συνάρτηση «**eval()**», αναμένεται ότι η εγγενής υποστήριξη του προτύπου **JSON** θα συμπεριληφθεί στην επόμενη έκδοση του προτύπου «**ECMAScript**».

2.9.3 Σχήμα JSON

Υπάρχουν πολλοί τρόποι για να γίνει επαλήθευση της δομής και των τύπων δεδομένων μέσα σε ένα αντικείμενο **JSON**, όπως με ένα σχήμα (**schema**) **XML**. Η προδιαγραφή «**JSON Schema**» είναι ένα πρότυπο βασισμένο στο **JSON** για τον προσδιορισμό της δομής των **JSON** δεδομένων. Το «**JSON Schema**» παρέχει μια σύμβαση για το τι δεδομένα **JSON** απαιτούνται για μια συγκεκριμένη εφαρμογή και με ποιο τρόπο μπορούν αυτά τα δεδομένα να τροποποιηθούν, ακριβώς ότι και το «**XML Schema**» προσφέρει για την **XML**. Επιπλέον, το «**JSON Schema**» έχει σχεδιαστεί να παρέχει έλεγχο εγκυρότητας, τεκμηρίωση και έλεγχο αλληλεπίδρασης των δεδομένων **JSON**. Βασίζεται στις ίδιες αρχές και ιδέες με αυτές των «**XML Schema**», «**RelaxNG**», και «**Kwalify**» αλλά έχει σχεδιαστεί να έχει ως βάση το **JSON**.

2.9.4 Χρήση του JSON στο AJAX

Ο κώδικας της **JavaScript** που ακολουθεί δείχνει πως ένας Πελάτης μπορεί να χρησιμοποιήσει μια αίτηση **XMLHttpRequest** για να ζητήσει ένα αντικείμενο σε μορφή **JSON** από τον Διακομιστή. (Ο προγραμματισμός στην πλευρά του Διακομιστή πρέπει να έχει στηθεί έτσι ώστε να απαντά στις αιτήσεις της διεύθυνσης «**url**» με μια συμβολοσειρά μορφοποιημένη σε **JSON**)

```
var the_object = {};
var http_request = new XMLHttpRequest();
http_request.open( "GET", url, true );
http_request.onreadystatechange = function () {
    if ( http_request.readyState == 4 && http_request.status ==
200 ) {
            the_object = JSON.parse( http_request.responseText
);
        }
};
http_request.send(null);
```

Να σημειωθεί ότι η χρήση του αντικειμένου **XMLHttpRequest** σε αυτό το παράδειγμα δεν είναι συμβατή με όλα τα προγράμματα περιήγησης. Υπάρχουν παραλλαγές στο συντακτικό για τους περιηγητές **Internet Explorer**, **Opera**, **Safari**, και σε όσους βασίζοντα στην τεχνολογία της **Mozilla**. Η χρησιμότητα του αντικειμένου **XMLHttpRequest** περιορίζεται από την πολιτική «ίδιας πηγής», η οποία ορίζει ότι η διεύθυνση **URL** της απάντησης προς την αίτηση πρέπει να βρίσκεται εντός του ίδιου τομέα **DNS** (**DNS domain**) με τον Διακομιστή που φιλοξενεί την σελίδα που δημιούργησε την αίτηση. Ως εναλλακτική λύση, η προσέγγιση «**JSONP**» ενσωματώνει τη χρήση μιας κωδικοποιημένης συνάρτησης επανάκλησης (**callback function**) η οποία κινείται μεταξύ του Πελάτη και του Διακομιστή και επιτρέπει στον Πελάτη να φορτώνει κωδικοποιημένα δεδομένα **JSON** από τομείς

«τρίτων» (**third-party domains**) και να γνωστοποιεί την συνάρτηση του καλούντος κατά την ολοκλήρωσή της. Ωστόσο, αυτή η λύση συνεπάγεται κάποιους κινδύνους ασφαλείας καθώς και επιπρόσθετες απαιτήσεις από την πλευρά του Διακομιστή.

Οι περιηγητές μπορούν επίσης να κάνουν χρήση των στοιχείων «**<iframe>**» για την ασύγχρονη αίτηση δεδομένων **JSON**, τρόπος ο οποίος είναι περισσότερο ανεξάρτητος προγράμματος περιήγησης, ή απλώς να κάνουν χρήση των υποβολών «**<form action="url_to_cgi_script" target="name_of_hidden_iframe">**». Αυτές οι προσεγγίσεις ήταν αρκετά διαδεδομένες πριν από την έλευση της ευρείας υποστήριξης του αντικειμένου **XMLHttpRequest**.

2.9.5 Θέματα Ασφαλείας

Παρόλο που το **JSON** προορίζεται ως ένα πρότυπο για σειριοποίηση δεδομένων (**data serialization format**), η σχεδίαση του ως ένα υποσύνολο της γλώσσας προγραμματισμού **JavaScript** δημιουργεί πολλές ανησυχίες γύρω από την ασφάλεια. Οι εν λόγω ανησυχίες επικεντρώνονταν στην χρήση ενός «διερμηνέα» της **JavaScript** (**JavaScript Interpreter**) για την δυναμική εκτέλεση κειμένου σε **JSON** ως κώδικα **JavaScript**, εκθέτοντας έτσι ένα πρόγραμμα σε κακόβουλο κώδικα που περιέχεται εκ των έσω του (γεγονός που αποτελεί και το κύριο μέλημα όταν πρόκειται για ανάκτηση δεδομένων μέσω του διαδικτύου). Αν και δεν είναι η μόνη μέθοδος για επεξεργασία του **JSON**, είναι μια απλή κα δημοφιλής τεχνική, η οποία απορρέει από την συμβατότητα του **JSON** με την συνάρτηση «**eval()**» της **JavaScript**, και η οποία απεικονίζεται στα ακόλουθα παραδείγματα κώδικα.

2.9.5.1 Η συνάρτηση «eval()» της JavaScript

Εξ' αιτίας του ότι κάθε κείμενο που έχει διαμορφωθεί σε πρότυπο **JSON** είναι επίσης και συντακτικά νόμιμος κώδικας της **JavaScript**, ένας εύκολος τρόπος για ένα πρόγραμμα της **JavaScript** να αναλύσει δεδομένα διαμορφωμένα σε **JSON** είναι η χρήση της ενσωματωμένης συνάρτησης «**eval()**», η οποία έχει σχεδιαστεί να αξιολογεί εκφράσεις σε **JavaScript** (**JavaScript expressions**). Αντί την χρήση ενός ειδικού αναλυτή του **JSON**, χρησιμοποιείται ο ίδιος ο διερμηνέας της **JavaScript** για την ανάλυση των δεδομένων **JSON** και την παραγωγή εγγενών αντικειμένων **JavaScript**.

Η τεχνική «**eval**» υπόκειται σε θέματα ευπαθούς ασφαλείας εάν τα δεδομένα και ολόκληρο το περιβάλλον της **JavaScript** δεν βρίσκεται υπό τον έλεγχο μιας και μοναδικής αξιόπιστης πηγής. Για παράδειγμα, αν τα δεδομένα τα ίδια δεν είναι έμπιστα, μπορεί να υπόκεινται σε επιθέσεις ενέσιμου κακόβουλου κώδικα **JavaScript** (**malicious code injection attacks**), εκτός εάν χρησιμοποιηθούν κάποια επιπλέον μέσα για επικύρωση των δεδομένων εξ' αρχής. «Τακτικές εκφράσεις» («**regular expressions**») χρησιμοποιούνται συχνά για να εκτελέσουν

αυτήν την επικύρωση προτού να κληθεί η συνάρτηση «**eval**». Επίσης, τέτοιες παραβιάσεις εμπιστοσύνης μπορεί να δημιουργήσουν τρωτά σημεία για κλοπή δεδομένων, πλαστογράφηση ταυτότητας, και άλλης ενδεχόμενης κακής χρήσης πόρων ή δεδομένων. Το πρότυπο **RFC** το οποίο ορίζει το **JSON** προτείνει την χρήση του παρακάτω κώδικα για την επικύρωση του **JSON** πριν την ανάλυση του (η μεταβλητή «**text**» είναι η είσοδος **JSON**):

```
var my_JSON_object = !(/[^,:{}\[\]0-9.\-+Eaeflnr-u \n\r\t]/.test(
text.replace(/"(\\.|[^"\\])*"/g, ''))) &&
eval('(' + text + ')');
```

Μια νέα συνάρτηση, η «**parseJSON()**», έχει προταθεί ως μια ασφαλέστερη εναλλακτική για την συνάρτηση «**eval**», καθώς είναι ειδικά κατασκευασμένη να επεξεργάζεται δεδομένα **JSON** και όχι δεδομένα **JavaScript**. Προοριζόταν να συμπεριληφθεί στην Τέταρτη Έκδοση του προτύπου **ECMAScript**, ωστόσο είναι σήμερα διαθέσιμη ως μια βιβλιοθήκη της **JavaScript** στην διεύθυνση http://www.JSON.org/json2.js και θα ενσωματωθεί στην Πέμπτη Έκδοση του **ECMAScript**.

2.9.5.2 Εγγενές JSON

Τα σύγχρονα προγράμματα περιήγησης πλέον παρέχουν, ή βρίσκονται στην φάση σχεδίασης για να μπορέσουν να παρέχουν, εγγενή υποστήριξη για κωδικοποίηση/αποκωδικοποιήση του **JSON**, γεγονός το οποίο τους απαλλάσσει από το πρόβλημα ασφάλειας με την χρήση της συνάρτησης «**eval**» που περιγράφεται παραπάνω. Επιπλέον, η εγγενής υποστήριξη του **JSON** παρέχει γενικότερα περισσότερη ταχύτητα σε σύγκριση με την χρήση των βιβλιοθηκών **JavaScript**. Από τον Ιούνιο του 2009, οι παρακάτω περιηγητές παρέχουν, ή πρόκειται στο άμεσο μέλλον να παρέχουν, εγγενή υποστήριξη για το πρότυπο **JSON**:

- **Mozilla Firefox 3.5+**
- **Microsoft Internet Explorer 8**
- Όσοι περιηγητές βασίζονται στην μηχανή **Webkit** (όπως ο **Safari** και ο **Google Chrome**)

Τουλάχιστον τέσσερις δημοφιλής βιβλιοθήκες της **JavaScript** προσφέρουν εγγενή υποστήριξη για το πρότυπο **JSON**:

- **Yahoo! UI Library**
- **jQuery**
- **Dojo Toolkit**
- **Mootools**

2.9.6 Σύγκριση με την XML

Η **XML** συχνά χρησιμοποιείται για να περιγράψει δομημένα δεδομένα και για να «σειριοποιήσει» αντικείμενα. Υπάρχουν διάφορα πρωτόκολλά που βασίζονται στην **XML** και αναπαριστούν το ίδιο είδος δομών δεδομένων με το πρότυπο **JSON** για τους ίδιους λόγους ανταλλαγής δεδομένων. Ωστόσο, επειδή η **XML** είναι μια γλώσσα σήμανσης γενικού τύπου, τα πρωτόκολλα αυτά είναι πιο περίπλοκα στο συντακτικό τους και μεγαλύτερα σε μέγεθος αρχείου από ότι το **JSON**, το οποίο σε αντίθεση με αυτά έχει σχεδιαστεί ειδικά για την ανταλλαγή δεδομένων.

Παρόλα τα παραπάνω, και τα δύο πρότυπα στερούνται ενός σαφούς μηχανισμού για την αναπαράσταση δυαδικού τύπου δεδομένων μεγάλου μεγέθους όπως είναι τα δεδομένα εικόνας (αν και τα δυαδικά δεδομένα μπορούν να σειριοποιηθούν και στις δυο περιπτώσεις με την εφαρμογή ενός γενικού τύπου σχεδίου για κωδικοποίηση δυαδικού-σε-κείμενο). Ένα επιπλέον μειονέκτημα του προτύπου **JSON,** είναι ότι στερείται της δυνατότητας παραπομπών (**references**) (κάτι που η **XML** διαθέτει μέσω επεκτάσεων όπως το «**XLink**» και το «**XPointer**») όπως επίσης και ενός προτύπου για τον συμβολισμό διαδρομής (**path notation**) συγκρίσιμο με το **XPath** της **XML**.

2.9.7 Αποδοτικότητα

Το πρότυπο **JSON** χρησιμοποιείται κυρίως για την επικοινωνία δεδομένων μέσω διαδικτύου, αλλά έχει και ορισμένα χαρακτηριστικά που μπορούν να περιορίσουν την αποδοτικότητά του για αυτόν τον σκοπό. Οι περισσότεροι περιορισμοί είναι γενικοί περιορισμοί των μορφών δεδομένων κειμένου και ισχύουν και για την **XML**. Για παράδειγμα, η αποκωδικοποίηση πρέπει να γίνεται σύμφωνα με την αρχή «χαρακτήρας-με-χαρακτήρας», και το πρότυπο δεν διαθέτει καμία πρόβλεψη για συμπίεση δεδομένων, «φυλάκιση» («**interning**») των συμβολοσειρών, ή παραπομπών σε αντικείμενα. Φυσικά, συμπίεση μπορεί να εφαρμοστεί σε δεδομένα που έχουν ήδη μορφοποιηθεί σε πρότυπο **JSON**. Στην πράξη, η επίδοση μπορεί να είναι συγκρίσιμη με αυτήν άλλων παρόμοιων προτύπων δεδομένων και συχνά εξαρτάται περισσότερο από την ποιότητα της υλοποίησης παρά από τους θεωρητικούς περιορισμούς των προτύπων.

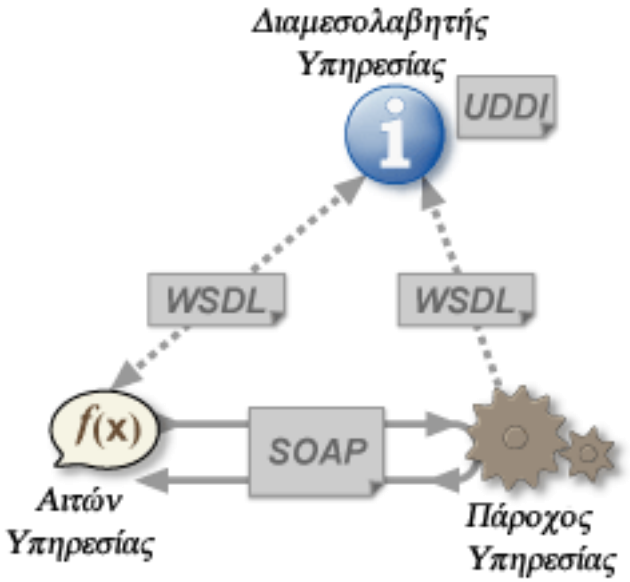
2.10 Διαδικτυακές Υπηρεσίες (Web Services)

2.10.1 Εισαγωγικά

Οι διαδικτυακές υπηρεσίες είναι συνήθως διεπαφές προγραμματισμού εφαρμογών (**Application Programming Interfaces** ή **APIs**) ή διαδικτυακές **APIs** (**Web APIs**) στις οποίες παρέχεται πρόσβαση μέσω ενός δικτύου, όπως είναι το διαδίκτυο, και οι οποίες εκτελούνται από ένα απομακρυσμένο σύστημα το οποίο και τις φιλοξενεί. Κοινώς, ο όρος «διαδικτυακές υπηρεσίες» αναφέρεται σε Πελάτες και Διακομιστές που επικοινωνούν μέσω ενός πρωτοκόλλου **HTTP** (**HyperText Transfer Protocol**) το οποίο χρησιμοποιείται στο διαδίκτυο. Τέτοιου είδους υπηρεσίες τείνουν να εμπίπτουν σε ένα από τα δύο ακόλουθα στρατόπεδα:

- «**Big Web Services**» («Μεγάλες Διαδικτυακές Υπηρεσίες»)
- «**RESTful Web Services**» («RESTful Διαδικτυακές Υπηρεσίες»)

Οι «**Big Web Services**» χρησιμοποιούν μηνύματα σε γλώσσα **XML** τα οποία ακολουθούν το πρότυπο «**SOAP**» (**Simple Object Access Protocol**) και έχουν γίνει αρκετά δημοφιλής στις παραδοσιακές εφαρμογές επιχειρήσεων. Σε τέτοια συστήματα, υπάρχει συχνά μια περιγραφή των λειτουργιών που προσφέρει η υπηρεσία, η οποία περιγραφή είναι γραμμένη σε γλώσσα **WSDL** (**Web Services Description Language**) και είναι αναγνώσιμη από μηχανή. Η χρήση της γλώσσας **WSDL** για την περιγραφή, δεν απαιτείται από ένα «τελικό σημείο» **SOAP** (**SOAP endpoint**), αλλά είναι απαραίτητη προϋπόθεση για την αυτοματοποιημένη παραγωγή του κώδικα μεταξύ Πελάτη-Διακομιστή από πολλά «πλαίσια» (**frameworks**) **SOAP** σε **Java** και **.NET** (αξιοσημείωτες εξαιρέσεις αποτελούν τα «πλαίσια» όπως **Spring**, **Apache Axis2** **Apache CXF**). Ορισμένες οργανώσεις της βιομηχανίας, όπως είναι η **WS-I**, χρήζουν ως υποχρεωτικά τόσο το **SOAP** όσο και την **WSDL** όταν ορίζουν μία διαδικτυακή υπηρεσία.



Πιο πρόσφατα, οι διαδικτυακές υπηρεσίες **REST** (**REpresentational State Transfer**) κερδίζουν δημοτικότητα, ιδιαίτερα στις διαδικτυακές εταιρίες. Διαδικτυακές υπηρεσίες **RESTfull** είναι οι υπηρεσίες που συμμορφώνονται με τους περιορισμούς των **REST**

υπηρεσιών. Χρησιμοποιώντας τις **HTTP** μεθόδους **PUT**, **GET** και **DELETE** παράλληλα με την **POST**, οι υπηρεσίες αυτές συχνά ενσωματώνονται καλύτερα με το πρωτόκολλο **HTTP** και τα προγράμματα περιήγησης διαδικτύου, από τις υπηρεσίες που βασίζονται στο **SOAP**. Αυτό συμβαίνει κυρίως επειδή δεν απαιτούν μηνύματα σε **XML** ή περιγραφές υπηρεσίας και **API** σε γλώσσα **WSDL.**

Η διαδικτυακή διεπαφή προγραμματισμού εφαρμογών (**Web API**) είναι μια εξέλιξη στις διαδικτυακές υπηρεσίες (σε ένα κίνημα που ονομάζεται «**Web 2.0**») όπου η έμφαση απομακρύνεται από τις υπηρεσίες που βασίζονται στο πρωτόκολλο **SOAP** και πηγαίνει προς το πιο άμεσο στυλ επικοινωνίας των **REST** υπηρεσιών. Οι **We APIs** επιτρέπουν των συνδυασμό των πολλαπλών διαδικτυακών υπηρεσιών σε νέες εφαρμογές οι οποίες είναι γνωστές ως «προσμίξεις» («**mashups**»).

Όταν χρησιμοποιείται στο πλαίσιο της ανάπτυξης ιστοσελίδων, η **Web API** είναι συνήθως ένα καθορισμένο σύνολο από μηνύματα αίτησης σε πρωτόκολλο **HTTP** μαζί με τον ορισμό της δομής των μηνυμάτων απάντησης, τα οποία συνήθως εκφράζονται σε μορφή **XML** ή **JSON**. Κατά την εκτέλεση σύνθετων υπηρεσιών διαδικτύου, κάθε επιμέρους υπηρεσία μπορεί να θεωρηθεί ως αυτόνομη. Ο χρήστης δεν έχει κανένα έλεγχο πάνω σε αυτές τις υπηρεσίες. Επιπλέον, οι ίδιες οι διαδικτυακές υπηρεσίες δεν είναι αξιόπιστες. Ο πάροχος της υπηρεσίας μπορεί να αφαιρέσει, αλλάξει ή αναβαθμίσει τις υπηρεσίες του χωρίς να ειδοποιήσει τους χρήστες. Η αξιοπιστία και η ανοχή σε βλάβες δεν υποστηρίζεται όσο θα έπρεπε, με αποτέλεσμα να υπάρχουν βλάβες ακόμα και κατά την διάρκεια εκτέλεσης μιας υπηρεσίας. Η διαχείριση των «εξαιρέσεων» (**exceptions**) στο πλαίσιο των διαδικτυακών υπηρεσιών είναι ακόμη ένα ανοικτό ερευνητικό θέμα.

Η κοινοπραξία **W3C** ορίζει μια «διαδικτυακή υπηρεσία» ως «ένα σύστημα λογισμικού σχεδιασμένο να υποστηρίζει διαλειτουργική αλληλεπίδραση μεταξύ μηχανών μέσω ενός δικτύου. Έχει μια διεπαφή που περιγράφεται μέσω ενός προτύπου το οποίο είναι επεξεργάσιμο από μηχανή (συγκεκριμένα, σε γλώσσα **WSDL**). Άλλα συστήματα αλληλεπιδρούν με την διαδικτυακή υπηρεσία με μια διαδικασία που προβλέπεται από την περιγραφή της χρησιμοποιώντας μηνύματα σε πρωτόκολλα **SOAP**, τα οποία συνήθως μεταδίδονται με την χρήση του πρωτοκόλλου **HTTP** με μια σειριοποίηση σε **XML** σε συνδυασμό με άλλα πρότυπα που είναι σχετικά με το διαδίκτυο.»

Η κοινοπραξία **W3C** αναφέρει επίσης ότι: «Μπορούμε να διακρίνουμε δυο μεγάλες κατηγορίες στις διαδικτυακές υπηρεσίες, τις **REST**-συμβατές Διαδικτυακές Υπηρεσίες, στις οποίες ο πρωταρχικός στόχος της υπηρεσίας είναι η διαχείριση **XML** αναπαραστάσεων των διαδικτυακών πόρων με την χρήση ενός ομοιόμορφου υποσυνόλου από λειτουργίες που δεν έχουν «καταστάσεις» (**stateless operation**), και τις αυθαίρετες Διαδικτυακές Υπηρεσίες, στις οποίες η υπηρεσία μπορεί να διαθέσει ένα αυθαίρετο υποσύνολο από λειτουργίες.»

2.10.2 Προδιαγραφές

2.10.2.1 Προφίλ

Για να βελτιώσει την διαλειτουργικότητα των διαδικτυακών εφαρμογών, η οργάνωση **WS-I** δημοσιεύει διάφορα «προφίλ». Ένα προφίλ είναι ένα σύνολο προδιαγραφών πυρήνα (**SOAP**, **WSDL**, …) σε μια συγκεκριμένη έκδοση (**SOAP 1.1**, **UDDI 2**, …) με ορισμένες επιπρόσθετες απαιτήσεις για τον περιορισμό της χρήσης των προδιαγραφών αυτών. Η οργάνωση **WS-I** δημοσιεύει επίσης και «περιπτώσεις χρήσης» (**use cases**) καθώς και εργαλεία δοκιμής (**test tools**) για να βοηθήσει την ανάπτυξη των διαδικτυακών υπηρεσιών που συμμορφώνονται με τα προφίλ της. Η «**WS**» είναι η «διαδικτυακή υπηρεσία» επεξεργασίας.

2.10.2.2 Πρόσθετες Προδιαγραφές, WS

Κάποιες προδιαγραφές έχουν αναπτυχθεί, ή βρίσκονται υπό ανάπτυξη, για την επέκταση των δυνατοτήτων των διαδικτυακών υπηρεσιών. Αυτές οι προδιαγραφές αναφέρονται γενικά ως «**WS-***». Ακολουθεί μια λίστα με ορισμένες από αυτές τις προδιαγραφές:

WS-Security

Προσδιορίζει τον τρόπο χρήσης «Κρυπτογράφησης σε **XML**» (**XML Encryption**) και «Υπογραφής **XML**» (**XML Signature**) στο πρωτόκολλο **SOAP** για ασφαλή ανταλλαγή μηνυμάτων, ως μια εναλλακτική ή επέκταση της χρήσης του **HTTPS** για την διασφάλιση του καναλιού μετάδοσης.

WS-Reliability

Ένα πρότυπο πρωτοκόλλου του οργανισμού **OASIS** (**Organization for the Advancement of Structured Information Standards**) για την αξιόπιστη ανταλλαγή μηνυμάτων μεταξύ δύο διαδικτυακών υπηρεσιών.

WS-Transaction

Ένας τρόπος διαχείρισης των συναλλαγών.

Ws-Addressing

Ένας καθιερωμένος τρόπος για την εισαγωγή διευθύνσεων στην «κεφαλίδα» (**header**) του πρωτοκόλλου **SOAP.**

Κάποιες από αυτές τις πρόσθετες προδιαγραφές έχουν προέλθει από την κοινοπραξία **W3C.** Γίνεται πολλή συζήτηση γύρω από την συμμετοχή της κοινοπραξίας, καθώς τα μοντέλα «καθολικού Ιστού» (**general Web**) και «Σημασιολογικού Ιστού» (**Semantic Web**) φαίνονται να βρίσκονται σε αντίθεση με ένα μεγάλο μέρος του οράματος για τις Διαδικτυακές Υπηρεσίες. Το γεγονός αυτό ήρθε πιο πρόσφατα στην επιφάνεια τον Φεβρουάριο του 2007,

στα πλαίσια της εκδήλωσης «**W3C Workshop on Web of Services for Enterprise Computing**», όπου κάποιοι από τους συμμετέχοντες υποστήριξαν την απόσυρση της κοινοπραξίας **W3C** από περεταίρω εργασίες πάνω στις προδιαγραφές «**WS-***» και την επικέντρωσή της στον πυρήνα του διαδικτύου. Αντίθετα, ο οργανισμός **OASIS** έχει τυποποιήσει αρκετές επεκτάσεις διαδικτυακής υπηρεσίας, συμπεριλαμβανομένων και των «**Web Services Resource Framework**» και «**WSDM**».

2.10.3 Μορφές της χρήσης

Οι διαδικτυακές υπηρεσίες είναι ένα σύνολο από εργαλεία τα οποία μπορούν να χρησιμοποιηθούν με πολλούς τρόπους. Οι τρεις πιο κοινές μορφές χρήσης τους είναι οι **RPC**, **SOA** και **REST**.

2.10.3.1 Κλήσεις Απομακρυσμένης Διαδικασίας (Remote Procedure Calls)

Οι διαδικτυακές υπηρεσίες τύπου **RPC** παρουσιάζουν μια κατανεμημένη λειτουργία (ή μέθοδο) διεπαφής της κλήσης η οποία είναι οικεία για πολλούς προγραμματιστές. Συνήθως, η βασική μονάδα των **RPC** διαδικτυακών υπηρεσιών είναι η λειτουργία **WSDL**.

Τα πρώτα εργαλεία των διαδικτυακών υπηρεσιών ήταν επικεντρωμένα σε **RPC**, και ως εκ τούτου αυτή η μορφή έχει αναπτυχθεί και υποστηριχτεί ευρέως. Ωστόσο, έχει δεχτεί αρκετές φορές κριτική επειδή δεν είναι «χαλαρής ζεύξης» (**loosely coupled**), διότι συχνά υλοποιήθηκε από υπηρεσίες χαρτογράφησης (**mapping services**) απευθείας σε κλήσεις μεθόδων ή συναρτήσεων που ήταν συγκεκριμένης γλώσσας. Πολλοί προμηθευτές αισθάνθηκαν ότι αυτή η προσέγγιση οδηγεί σε αδιέξοδο, και πίεσαν για την απαλοιφή των **RPC** από το **Βασικό Προφίλ WS-I** (**WS-I Basic Profile**).

Άλλες προσεγγίσεις με σχεδόν την ίδια λειτουργικότητα με των **RPC** είναι η αρχιτεκτονική **CORBA** (**Common Object Request Broker Architecture**) του γκρουπ **OMG** (**Object Management Group**), το μοντέλο **DCOM** (**Distributed Object Model**) της **Microsoft** και η μέθοδος απομακρυσμένη επίκληση της **Java** (**Java Remote Method Invocation**) της **Sun Microsystems**.

2.10.3.2 Υπηρεσιοστραφής Αρχιτεκτονική (Service-oriented Architecture)

Οι διαδικτυακές υπηρεσίες μπορούν επίσης να χρησιμοποιηθούν για την υλοποίηση μιας αρχιτεκτονικής σύμφωνα με το πρότυπο της **SOA** (**Service-oriented Architecture**), όπου η βασική μονάδα της επικοινωνίας είναι ένα μήνυμα, και όχι μια λειτουργία. Αυτό συχνά αναφέρεται και ως μηνυματοστραφείς υπηρεσίες (**«message-oriented» services**).

Οι διαδικτυακές υπηρεσίες που ακολουθούν την αρχιτεκτονική **SOA** υποστηρίζονται από τους κυριότερους προμηθευτές λογισμικού και αναλυτές του κλάδου. Σε αντίθεση με τις

υπηρεσίες των **RPC**, η «χαλαρή ζεύξη» είναι πιο πιθανή, επειδή η επικέντρωση είναι στην «σύμβαση» που παρέχει η **WSDL** και όχι στις υποκείμενες λεπτομέρειες τις υλοποίησης. Οι αναλυτές του «ενδιάμεσου λογισμικού» (**middleware**) χρησιμοποιούν Επιχειρησιακούς Διαύλους Υπηρεσίας (**Enterprise Service Buses** ή **ESB**) οι οποίοι συνδυάζουν «μηνυματοστραφή» επεξεργασία και Διαδικτυακές Υπηρεσίες για την δημιουργία μιας αρχιτεκτονικής **SOA** που περιστρέφεται γύρω από συμβάντα (**Event-driven SOA**). Δυο παραδείγματα αρχιτεκτονικών **ESB** ανοικτού κώδικα είναι τα «**Mule**» και «**Open ESB**».

2.10.3.3 Μεταφορά Αντιπροσωπευτικών Καταστάσεων (REpresentational State Transfer ή REST)

Η μεταφορά **REST** επιχειρεί να περιγράψει τις αρχιτεκτονικές που κάνουν χρήση του πρωτοκόλλου **HTTP** ή παρόμοιων πρωτοκόλλων, περιορίζοντας την διεπαφή σε ένα σύνολο από καλά γνωστές και καθιερωμένες λειτουργίες (όπως οι **GET**, **POST**, **PUT**, **DELETE** για το **HTTP**). Εδώ η προσοχή εστιάζεται στην αλληλεπίδραση με «**stateful**» πόρους, παρά με μηνύματα ή λειτουργίες.

Μια αρχιτεκτονική που βασίζεται στην **REST** (δηλαδή που είναι «**RESTful**») μπορεί να χρησιμοποιήσει την γλώσσα **WSDL** για την περιγραφή **SOAP** μηνυμάτων μέσω πρωτοκόλλου **HTTP**, μπορεί να υλοποιηθεί ως καθαρά αφηρημένη πάνω από το **SOAP** (π.χ. η **WS-Transfer**), ή μπορεί να υλοποιηθεί εξ' ολοκλήρου χωρίς την χρήση του **SOAP**.

Η έκδοση **2.0** της **WSDL** παρέχει υποστήριξη για δέσμευση σε όλες τις μεθόδους αίτησης του **HTTP** (όχι μόνο για την **GET** και **POST** όπως στην έκδοση **1.1**) και έτσι επιτρέπει μια καλύτερη υλοποίηση των Διαδικτυακών Υπηρεσιών **RESTful**. Ωστόσο, η υποστήριξη για αυτήν τη προδιαγραφή είναι ακόμα φτωχή στα πακέτα ανάπτυξης λογισμικού, τα οποία συχνά προσφέρουν εργαλεία μόνο για την έκδοση **1.1** της **WDSL**.

2.10.4 Μεθοδολογίες Σχεδιασμού

Οι διαδικτυακές υπηρεσίες μπορούν αν αναπτυχθούν με δύο τρόπους:

- Χρησιμοποιώντας την μέθοδο «από κάτω προς τα πάνω» (**bottom up method**) ένας προγραμματιστής γράφει πρώτα την κλάση υλοποίησης σε μια προγραμματιστική γλώσσα, και στην συνέχεια χρησιμοποιεί ένα εργαλείο παραγωγής κώδικα **WSDL** για να εκθέσει τις μεθόδους της ως μια διαδικτυακή υπηρεσία. Αυτή συχνά είναι και η πιο εύκολη προσέγγιση.
- Χρησιμοποιώντας την μέθοδο «από πάνω προς τα κάτω», ένας προγραμματιστής γράφει πρώτα το έγγραφο της **WSDL** και μετά χρησιμοποιεί ένα εργαλείο παραγωγής κώδικα για να παραγάγει μια κλάση σκελετό, την οποία και συμπληρώνει

στην συνέχεια. Αυτός ο τρόπος γενικά θεωρείται πιο δύσκολος αλλά μπορεί να παράγει «καθαρότερες» σχεδιάσεις.

2.10.5 Κριτικές

Οι επικριτές των μη **RESTful** διαδικτυακών υπηρεσιών συχνά παραπονιούνται ότι είναι υπερβολικά περίπλοκες και βασίζονται πάνω σε μεγάλους προμηθευτές λογισμικού, και όχι σε τυπικές υλοποιήσεις ανοικτού κώδικα. Υπάρχουν όμως υλοποιήσεις ανοικτού κώδικα, όπως είναι οι «**Apache Axis**» και «**Apache CXF**».

Στην αντίπερα όχθη, μια βασική ανησυχία των προγραμματιστών των **REST** διαδικτυακών υπηρεσιών είναι το γεγονός ότι τα πακέτα εργαλείων διαδικτυακών υπηρεσιών **SOAP** κάνουν τον καθορισμό νέων διεπαφών για απομακρυσμένη αλληλεπίδραση πολύ εύκολο, γιατί συχνά στηρίζονται στην «ενδοσκόπηση» για να εξαγάγουν την **WSDL** και την **API** της υπηρεσίας από κώδικα σε **Java**, **C#** ή **VB**. Υποστηρίζεται ότι αυτό μπορεί να αυξήσει την ευθραυστότητα των συστημάτων, εφόσον μια μικρή αλλαγή στον Διακομιστή (ακόμη και μια αναβάθμιση της στοίβας του **SOAP**) μπορεί να έχει ως αποτέλεσμα μια διαφορετική **WSDL** και μια διαφορετική διεπαφή της υπηρεσίας. Οι κλάσεις στην πλευρά του Πελάτη που μπορούν να παραχθούν από τις **WSDL** και **XSD** περιγραφές της υπηρεσίας είναι συχνά παρόμοια συνδεδεμένες με μια συγκεκριμένη έκδοση του «τελικού σημείου» **SOAP** (**SOAP endpoint**) και μπορούν να «σπάσουν» εάν το «τελικό σημείο» αλλάξει ή εάν αναβαθμιστεί η στοίβα του **SOAP** στην πλευρά του Πελάτη. Καλά σχεδιασμένα «τελικά σημεία» του **SOAP** (με χειρόγραφες **WSDL** και **XSD**) δεν «πάσχουν» από αυτό το πρόβλημα, αλλά υπάρχει ακόμα το πρόβλημα κατά το οποίο μια προσαρμοσμένη διεπαφή για κάθε υπηρεσία απαιτεί και έναν προσαρμοσμένο Πελάτη για κάθε υπηρεσία.

Υπάρχουν επίσης ανησυχίες σχετικά με τις επιδόσεις που οφείλονται στο ότι οι διαδικτυακές υπηρεσίες κάνουν χρήση της **XML** ως πρότυπο του μηνύματος και των **SOAP**/**HTTP** για την περίκλειση (**enveloping**) και την μεταφορά. Ωστόσο, αναδυόμενες τεχνολογίες για την ανάλυση και δημιουργία ευρετηρίων της **XML**, όπως είναι η **VTD-XML**, υπόσχονται να αντιμετωπίσουν τα θέματα επιδόσεων που σχετίζονται με την **XML**.

2.11 Η πλατφόρμα ανάπτυξης εφαρμογών ColdFusion

2.11.1 Τι είναι το ColdFusion;

Το **ColdFusion** είναι μια εμπορική πλατφόρμα ταχείας ανάπτυξης λογισμικού (**rapid application development platform**) η οποία εφευρέθηκε από τον **Jeremy** και τον **JJ Alleire** το 1995. Αρχικά είχε σχεδιαστεί για να διευκολύνει την σύνδεση απλών σελίδων **HTML** με

μια βάση δεδομένων, όμως από την έκδοση **2** είχε γίνει μια πλήρης πλατφόρμα η οποία περιελάμβανε ένα **Ολοκληρωμένο Περιβάλλον Ανάπτυξης** (**Intergraded Development Environment** ή **IDE**) και μια πλήρης γλώσσα προγραμματισμού «σεναρίων» «**scripting language**». Στις τρέχουσες εκδόσεις του **ColdFusion**, οι οποίες πωλούνται από την **Adobe Systems**, περιλαμβάνονται και προηγμένες δυνατότητες για την ανάπτυξη και την ενσωμάτωση «πλούσιων» διαδικτυακών εφαρμογών (**rich internet applications**) από επιχειρήσεις. Το **ColdFusion** ανταγωνίζεται κατά κύριο λόγο τις γλώσσες **PHP** και **ASP**.

2.11.2 Γενική Επισκόπηση

Ένα από τα χαρακτηριστικά του **ColdFusion** που το κάνουν να ξεχωρίζει, είναι η γλώσσα προγραμματισμού «σεναρίων» που το συνοδεύει, ή **CFML** (**ColdFusion Markup Language**), η οποία ανταγωνίζεται τα «εξαρτήματα σεναρίων» (**scripting components**) των γλωσσών **ASP**, **JSP** και **PHP** στον σκοπό και στα χαρακτηριστικά, μολονότι μοιάζει περισσότερο με την γλώσσα **HTML** όσον αφορά το συντακτικό. Ο όρος «**ColdFusion**» συχνά χρησιμοποιείται ως συνώνυμος του όρου «**CFML**», όμως υπάρχουν και άλλοι Διακομιστές εφαρμογών που χρησιμοποιούν την **CFML** εκτός από τον **ColdFusion**. Επιπλέον, το **ColdFusion** υποστηρίζει και άλλες γλώσσες προγραμματισμού πέραν της **CFML**, όπως είναι η γλώσσα **ActionScript** καθώς και ενσωματωμένα «σενάρια» (**scripts**) γραμμένα σε γλώσσα **CFScript** (μια γλώσσα που μοιάζει με την **JavaScript**).

Αρχικά το **ColdFusion** αναπτύχθηκε από τους αδερφούς **Allaire** και κυκλοφόρησε τον Ιούλιο του 1995 από την εταιρία τους, την «**Allaire**». Το 2001 η εταιρία **Allaire** αποκτήθηκε από την εταιρία **Macromedia**, η οποία με την σειρά της αποκτήθηκε από την **Adobe Systems Inc.** το 2005. Η πλατφόρμα **ColdFusion** χρησιμοποιείται κατά κύριο λόγο σε ιστοσελίδες και δίκτυα «**intranets**» που βασίζονται πολύ σε δεδομένα, μπορεί όμως να χρησιμοποιηθεί και για την παραγωγή απομακρυσμένων υπηρεσιών όπως οι διαδικτυακές υπηρεσίες «**SOAP**» και το «**Flash remoting**». Επίσης, είναι σχεδιασμένο να λειτουργεί αρμονικά ως η τεχνολογία στην πλευρά του Διακομιστή για την τεχνολογία **Flex** της πλευράς του Πελάτη.

2.11.2.1 Κύρια Χαρακτηριστικά

Το **ColdFusion** παρέχει μια σειρά από επιπρόσθετα χαρακτηριστικά, τα οποία προσφέρονται με την βασική του εγκατάσταση. Μεταξύ αυτών είναι τα παρακάτω:

- Απλοποιημένη πρόσβαση σε βάσεις δεδομένων
- Διαχείριση προσωρινής μνήμης «**cache**» μεταξύ Πελάτη και Διακομιστή
- Παραγωγή του κώδικα μεταξύ Πελάτη - Διακομιστή, ειδικά όσον αφορά την επικύρωση και τα «βοηθητικά προγράμματα» (**widgets**) των φορμών.
- Μετατροπή από την **HTML** σε **PDF** και **FlashPaper**

- Ανάκτηση δεδομένων από κοινά επιχειρηματικά συστήματα όπως τα **Active Directory**, **LDAP**, **SMTP**, **POP**, **HTTP**, **FTP**, **Microsoft Exchange Server** και από κοινά πρότυπα δεδομένων όπως **RSS** και **Atom**
- Υπηρεσία δημιουργίας ευρετηρίου και αναζήτησης αρχείων, βασισμένη στο «**Verity K2**»
- Διαχείριση Γραφικής Διεπαφής Χρήστη (**GUI Administration**)
- Πεδία (**scopes**) Διακομιστή, Πελάτη, εφαρμογής, συνόδου και αίτησης
- Ανάλυση, δημιουργία ερωτημάτων (**XPath**), επικύρωση και μετασχηματισμός (**XSLT**) του κώδικα της **XML**
- Συσταδοποίηση Διακομιστή (**Server clustering**)
- Προγραμματισμός εργασιών
- Δημιουργία αναφορών και γραφικών παραστάσεων
- Απλοποιημένος χειρισμός αρχείων, που περιλαμβάνει γραφικά ράστερ (και **CAPTCHA**) και συμπιεσμένα αρχεία «**zip**» (σε μελλοντική έκδοση σχεδιάζεται η δυνατότητα χειρισμού και του βίντεο)
- Απλοποιημένη υλοποίηση διαδικτυακής υπηρεσίας (με αυτοματοποιημένη παραγωγή κώδικα **WSDL** / διαφανή χειρισμό του πρωτοκόλλου **SOAP** για την δημιουργία και καταστροφή υπηρεσιών)

Άλλες υλοποιήσεις της γλώσσας **CFML** προσφέρουν παρόμοιες ή ενισχυμένες λειτουργίες, όπως η εκτέλεση μέσα σε ένα περιβάλλον «**.NET**» ή ο χειρισμός εικόνων.

Η μηχανή της πλατφόρμας έχει γραφτεί σε γλώσσα **C** και περιλαμβάνει, μεταξύ άλλων, μια ενσωματωμένη γλώσσα «σεναρίων» (την **CFScript**), επιπρόσθετα υποσυστήματα (**plug-in modules**) γραμμένα σε **Java**, και ένα συντακτικό παρόμοιο με αυτό της **HTML**. Μία ετικέτα (**tag**) του **ColdFusion**, το ισοδύναμο ενός στοιχείου της **HTML**, ξεκινάει με τα γράμματα «**CF**» ακολουθούμενα από ένα όνομα το οποίο είναι ενδεικτικό του πως αυτή η ετικέτα ερμηνεύεται στην **HTML**. Για παράδειγμα η ετικέτα «**<cfoutput>**» η οποία ξεκινάει την έξοδο μεταβλητών ή άλλου περιεχομένου.

Εκτός της **CFScript** και των επιπρόσθετων προγραμμάτων (**plug-ins**), παρέχεται και η σχεδιαστική πλατφόρμα **CFStudio** η οποία ακολουθεί το σύστημα **WYSIWYG** (**What You See Is What You Get**). Το **CFStudio** υποστηρίζει συντακτικό από δημοφιλείς γλώσσες για «**backend**» προγραμματισμό, όπως είναι η **Perl**. Για να ενισχύσει την ευκολότερη δημιουργία των «**backend**» λειτουργιών από μη προγραμματιστές, από την έκδοση **4.0** και πέρα, το **CFStudio** παρέχει ευκολότερη ενσωμάτωση με τους Διακομιστές **Apache Web Server** και **Internet Information Server**.

2.11.2.2 Άλλα Χαρακτηριστικά

Η πρώτη έκδοση του **ColdFusion** (τότε ονομαζόταν **Cold Fusion**) κυκλοφόρησε τον Ιούλιο του 1995. Οι πρώτες εκδόσεις ήταν σχεδόν ολοκληρωτικά γραμμένες από ένα άτομο, τον **Joseph JJ Allaire**. Πρωτόγονές μπροστά στα σύγχρονα πρότυπα, οι πρώτες εκδόσεις του **ColdFusion** πρόσφεραν κάτι περισσότερο από μια απλή πρόσβαση σε βάσεις δεδομένων. Όλες οι εκδόσεις του **ColdFusion** πριν της έκτης ήταν γραμμένες στην γλώσσα **Visual C++** της **Microsoft**. Αυτό σήμαινε ότι το **ColdFusion** ήταν περιορισμένο να λειτουργεί σε περιβάλλον των **Microsoft Windows**, αν και η εταιρία **Allaire** μετέφερε με επιτυχία το **ColdFusion** στο λειτουργικό σύστημα **Sun Solaris** από την έκδοση **3.1** και πέρα.

Για λόγους που μάλλον συνδέονται με περιορισμένες πωλήσεις, η εταιρία εξαγοράστηκε από την **Macromedia**, και στην συνέχεια από την **Adobe Systems**. Από την έκδοση **4.0** και μετά, το **ColdFusion** έγινε περισσότερο δυνατό και εύρωστο. Με την κυκλοφορία του **ColdFusion MX 6**, η μηχανή της πλατφόρμας γράφτηκε εκ νέου σε γλώσσα **Java** και υποστήριζε το δικό της περιβάλλον εκτέλεσης (**runtime environment**). Η έκδοση **6.1** παρουσίασε για πρώτη φορά την δυνατότητα συγγραφής και εντοπισμού σφαλμάτων κώδικα **Shockwave Flash**.

2.11.2.3 Adobe ColdFusion Builder

Το **ColdFusion Builder** (με κωδικό όνομα «**Bolt**») είναι το νέο **Ολοκληρωμένο Περιβάλλον Ανάπτυξης** (**Intergraded Development Environment** ή **IDE**) της **Adobe**, το οποίο βασίζεται στο περιβάλλον **Eclipse**, και μπορεί να χρησιμοποιηθεί για την κατασκευή εφαρμογών για το **ColdFusion**. Η κωδική ονομασία «**Bolt**» προέρχεται από το αρχικό εικονίδιο αστραπής που είχε το προϊόν από τον καιρό που το διένειμε η εταιρία **Allaire**. Το **ColdFusion Builder** διένυσε μια μεγάλη περίοδο δοκιμαστικής λειτουργίας και τελικά δόθηκε στην κυκλοφορία τον Μάρτιο του 2010 την ίδια μέρα με την κυκλοφορία του **Flash Builder 4**. Μερικά από τα χαρακτηριστικά του είναι τα ακόλουθα:

- Αυτόματη ρύθμιση της Σχεσιακής Απεικόνισης Αντικειμένων (**Object Relational Mapping**)
- Παραγωγή Κώδικα Εφαρμογής (**Application Code Generation**)
- Διαχείριση του Διακομιστή
- Εύκολη επέκταση μέσω του «πλαισίου» (**framework**) **Eclipse**
- Απόδοση Έμφασης στο συντακτικό (**syntax highlighting**) των **CFML**, **HTML**, **JavaScript** και **CSS**
- Βοήθεια στην συγγραφή του κώδικα (**code assist**) για ετικέτες, συναρτήσεις, μεταβλητές και εξαρτήματα (**components**)
- Αναδίπλωση Κώδικα

- Δημιουργία και διαχείριση «αποσπασμάτων κώδικα» (**snippets**)
- Προβολή Περιγράμματος
- Απομακρυσμένος εξερευνητής για αρχεία και βάσεις δεδομένων
- Αποσφαλμάτωση ανά επίπεδο γραμμής (**Line-level Debugging**)
- Απλοποίηση δομής κώδικα (**Refactoring**)

2.11.3 Ανάλυση Επιλεγμένων Χαρακτηριστικών

2.11.3.1 Πλούσιες Φόρμες (Rich Forms)

Ο Διακομιστής **ColdFusion Server** περιλαμβάνει ένα υποσύνολο της τεχνολογίας **Macromedia Flex 1.5**. Δεδηλωμένος στόχος της είναι καταστεί δυνατή η δημιουργία πλούσιων φορμών σε σελίδες **HTML**, χρησιμοποιώντας την γλώσσα **CFML** για την παραγωγή ταινιών **Flash**. Αυτές οι φόρμες σε **Flash** μπορούν να χρησιμοποιηθούν για την ενσωμάτωση πλούσιων διαδικτυακών εφαρμογών, αλλά με περιορισμένη αποδοτικότητα λόγω των περιορισμών του **ActionScript** που έχουν αποδοθεί σε φόρμες **Flash** από την **Macromedia**.

Οι φόρμες σε **Flash** παρέχουν επιπλέον βοηθητικά προγράμματα (**widgets**) για εισαγωγή δεδομένων, όπως είναι ο «επιλογέας ημερομηνιών» και τα «πλέγματα δεδομένων» (**data grids**). Σε προηγούμενες εκδόσεις του **ColdFusion**, ήταν διαθέσιμα κάποια επιπρόσθετα «βοηθητικά προγράμματα» (**widgets**) και μια μερική επικύρωση φορμών, χρησιμοποιώντας ένα συνδυασμό από βοηθητικές εφαρμογές σε **Java** (**Java applets**) και της γλώσσας **JavaScript**. Αυτή η επιλογή εξακολουθεί να ισχύει για όσους δεν επιθυμούν να χρησιμοποιήσουν **Flash**, αν και δεν διαθέτει πλήρη υποστήριξη χαρακτηριστικών.

Ένα παράδειγμα πλούσιας φόρμας με χρήση **Flash**, είναι το ακόλουθο:

```
<cfform format="flash" method="post" width="400" height="400">
    <cfinput type="text" name="username" label="Username"
required="yes" >
    <cfinput type="password" name="password" label="Password"
required="yes" >
    <cfinput type="submit" name="submit" value="Sign In" >
</cfform>
```

Το ColdFusion περιλαμβάνει επίσης και δυνατότητα δημιουργίας «**XForms**» και την δυνατότητα να «ντύσει» φόρμες χρησιμοποιώντας την γλώσσα μετασχηματισμού **XSLT**.

2.11.3.2 Παραγωγή εγγράφων PDF και FlashPaper

Το **ColdFusion** μπορεί να παράξει έγγραφα **PDF** ή **FlashPaper** χρησιμοποιώντας τυπική **HTML** (δηλαδή, δεν χρειάζεται επιπλέον προγραμματισμό για να παράξει έγγραφα προς

εκτύπωση). Οι συντάκτες της **CFML** απλά τοποθετούν τον κώδικα **HTML** και **CSS** μέσα σε ένα ζευγάρι από ετικέτες «**cfdocument**» και προσδιορίζουν το επιθυμητό πρότυπο (**PDF** ή **FlashPaper**). Το παραγόμενο έγγραφο μπορεί να είτε να αποθηκευτεί στον δίσκο είτε να αποσταλεί στον περιηγητή του Πελάτη. Η έκδοση **8** του **ColdFusion** εισήγαγε την ετικέτα «**cfpdf**» η οποία επιτρέπει έναν πρωτοφανή έλεγχο των εγγράφων **PDF** συμπεριλαμβανομένων των **PDF** φορμών και της συγχώνευσης των **PDFs.** Ωστόσο, αυτές οι ετικέτες δεν κάνουν χρήση της μηχανής **PDF** της **Adobe**, αλλά μιας δωρεάν και ανοικτού κώδικα βιβλιοθήκης της **Java**, η οποία ονομάζεται «**iText**».

2.11.3.3 Τα εξαρτήματα (components) του ColdFusion

Αρχικά το **ColdFusion** δεν ήταν μια αντικειμενοστραφής προγραμματιστική πλατφόρμα, και μέχρι σήμερα στερείται μερικών αντικειμενοστραφών χαρακτηριστικών. Το **ColdFusion** ανήκει στην κατηγορία των αντικειμενοστραφών γλωσσών που δεν υποστηρίζουν πολλαπλή κληρονομικότητα (μαζί με την **Java**, **Smalltalk**, κ.α.). Με την κυκλοφορία της έκδοσης **MX** (**6+**), το **ColdFusion** εισήγαγε την κατασκευή «εξαρτημάτων» (**components**) τα οποία παρομοιάζονται με τις κλάσεις στις αντικειμενοστραφείς γλώσσες. Κάθε «**εξάρτημα**» μπορεί να περιέχει οποιονδήποτε αριθμό από ιδιότητες (**properties**) και μεθόδους. Ένα «**εξάρτημα**» μπορεί επίσης να επεκτείνει (**extend**) κάποιο άλλο (κληρονομικότητα). Τα «εξαρτήματα» υποστηρίζουν μόνο μονή κληρονομικότητα. Με την έκδοση **8**, το **ColdFusion** υποστηρίζει πλέον και διεπαφές στο στυλ της **Java**. Τα «εξαρτήματα» του **ColdFusion** χρησιμοποιούν την κατάληξη αρχείου «**.cfc**» για να διαφοροποιούνται από τα πρότυπα (**templates**) του **ColdFusion** που χρησιμοποιούν την κατάληξη «**.cfm**».

2.11.3.4 Remoting

Οι μέθοδοι των «εξαρτημάτων» μπορούν να γίνουν διαθέσιμες ως διαδικτυακές υπηρεσίες χωρίς την ανάγκη επιπλέον προγραμματισμού και ρυθμίσεων. Το μόνο που απαιτείται για την πρόσβαση σε μία μέθοδο είναι η δήλωσή της ως «**remote**» («απομακρυσμένη»). Το ColdFusion αυτόματα παράγει μια περιγραφή σε γλώσσα **WSDL** (**Web Services Description Language**) στην διεύθυνση **URL** του «εξαρτήματος» με αυτόν τον τρόπο:

http://διαδρομή/προς/εξαρτήματα/Εξάρτημα.cfc?wsdl

Εκτός από το πρότυπο «**SOAP**», οι υπηρεσίες προσφέρονται και στο δυαδικό πρότυπο «**Flash Remoting**».

Οι μέθοδοι που έχουν δηλωθεί ως «**remote**» μπορούν επίσης να κληθούν μέσω μιας αίτησης «**HTTP GET**» ή «**HTTP POST**». Για παράδειγμα, δίδεται η αίτηση «**GET**» που ακολουθεί:

```
http://path/to/components/Component.cfc?method=search&query=your+query&mode=st
rict
```

Αυτή η αίτηση θα καλέσει την συνάρτηση αναζήτησης του εξαρτήματος, και θα περάσει ως ορίσματα τα «**your query**» και «**strict**». Αυτού του είδους οι επικλήσεις είναι κατάλληλες για εφαρμογές που κάνουν χρήση **AJAX**. Η έκδοση **8** του **ColdFusion** εισήγαγε την δυνατότητα της σειριοποίησης δομών δεδομένων του **ColdFusion** σε μορφή **JSON** για χρήση από την πλευρά του Πελάτη.

Ο Διακομιστής του **ColdFusion** θα παράξει αυτόματα την τεκμηρίωση (**documentation**) για ένα «εξάρτημα», εάν περιηγηθείτε στην διεύθυνση **URL** του και εισάγετε τον κατάλληλο κώδικα μέσα στις δηλώσεις (**declarations**) του «εξαρτήματος». Αυτή είναι μια εφαρμογή της «ενδοσκόπησης» ενός «εξαρτήματος», η οποία διατίθεται στους προγραμματιστές των «εξαρτημάτων» του **ColdFusion**. Η πρόσβαση στην τεκμηρίωση ενός «εξαρτήματος» απαιτεί έναν κωδικό πρόσβασης. Ένας προγραμματιστής μπορεί να προβάλει την τεκμηρίωση για όλα τα «εξαρτήματα» που είναι γνωστά στον Διακομιστή του **ColdFusion** με την πλοήγηση στην διεύθυνση **URL** του **ColdFusion**. Αυτή η διεπαφή μοιάζει με αυτήν της **HTML** τεκμηρίωσης **Javadoc** (**Javadoc HTML Documentation**) για τις κλάσεις της **Java**.

2.11.3.5 Προσαρμοσμένες Ετικέτες (Custom Tags)

Το **ColdFusion** παρέχει διάφορους τρόπους για την ενσωμάτωση προσαρμοσμένων ετικετών σε γλώσσα σήμανσης, δηλαδή εκείνων που δεν περιλαμβάνονται στην γλώσσα του πυρήνα του **ColdFusion**. Οι προσαρμοσμένες ετικέτες είναι ιδιαίτερα χρήσιμες στο να παρέχουν μία οικεία διεπαφή για τους προγραμματιστές διαδικτύου και τους συντάκτες περιεχομένου που είναι εξοικειωμένοι με την **HTML** αλλά όχι με τον «επιτακτικό προγραμματισμό» (**imperative programming**).

Ο παραδοσιακός και πιο συνηθισμένος τρόπος είναι με την χρήση της **CFML**. Μια βασική **CFML** σελίδα μπορεί να ερμηνευτεί ως μια ετικέτα, με το όνομα της ετικέτας να αντιστοιχεί στο όνομα του αρχείου με το πρόθεμα «**cf_**». Για παράδειγμα, το αρχείο «**IMAP.cfm**» μπορεί να χρησιμοποιηθεί ως η ετικέτα «**cf_imap**». Οι παράμετροι που χρησιμοποιούνται μέσα στην ετικέτα είναι διαθέσιμοι στο πεδίο «**ATTRIBUTES**» της σελίδας υλοποίησης της ετικέτας. Οι σελίδες **CFML** είναι προσβάσιμες από τον ίδιο κατάλογο με αυτόν της καλούσας σελίδας, μέσω ενός ειδικού καταλόγου της διαδικτυακής εφαρμογής του **ColdFusion**, ή μέσω μιας ετικέτας «**CFIMPORT**» μέσα από την καλούσα σελίδα. Η τελευταία μέθοδος δεν απαιτεί κατ' ανάγκη το πρόθεμα «**cf_**» για το όνομα της ετικέτας.

Ένας δεύτερος τρόπος είναι η ανάπτυξη των «**CFX**» ετικετών χρησιμοποιώντας **Java** ή **C++**. Οι ετικέτες «**CFX**» έχουν το πρόθεμα «**cfx_**», για παράδειγμα «**cfx_imap**». Οι ετικέτες

προστίθενται στο περιβάλλον εκτέλεσης του **ColdFusion** χρησιμοποιώντας τον Διαχειριστή **ColdFusion** (**ColdFusion Administrator**), όπου καταχωρούνται αρχεία «**JAR**» ή «**DLL**» ως προσαρμοσμένες ετικέτες. Τέλος, το **ColdFusion** υποστηρίζει βιβλιοθήκες ετικέτας **JSP** από την προδιαγραφή **2.0** της γλώσσας **JSP**. Οι ετικέτες **JSP** μπορούν να συμπεριληφθούν σε σελίδες **CFML** χρησιμοποιώντας την ετικέτα «**CFIMPORT**».

2.11.3.6 Εναλλακτικά περιβάλλοντα Διακομιστή

Το **ColdFusion** ξεκίνησε ως μια ιδιόκτητη τεχνολογία, που ήταν βασισμένη σε βιομηχανικά πρότυπα Διαδικτυακής τεχνολογίας. Ωστόσο, γίνεται όλο και λιγότερο κλειστή τεχνολογία μέσω της διαθεσιμότητας από ανταγωνιστικά προϊόντα. Τα προϊόντα αυτά περιλαμβάνουν τις τεχνολογίες **Railo**, **BlueDragon**, **IgniteFusion**, **SmithProject** και **Coral Web Builder**.

Μπορεί εύκολα να στηριχθεί το επιχείρημα που φέρει το **ColdFusion** να είναι λιγότερο δεσμευτικό πλατφόρμας από τις αντίπαλες τεχνολογίες **J2EE** ή **.NET**, απλά επειδή το **ColdFusion** μπορεί να εκτελεστεί πάνω από έναν Διακομιστή εφαρμογών **.NET**, ή πάνω από οποιονδήποτε κοντέινερ ενός «**servlet**» ή Διακομιστή εφαρμογών **J2EE** (όπως **JRun**, **WebSphere**, **JBoss**, **Geronimo**, **Tomcat**, **Resin Server**, **Jetty web server**, κ.τ.λ.). Θεωρητικά, μια εφαρμογή **ColdFusion** μπορεί να μεταφερθεί χωρίς αλλαγές από έναν Διακομιστή εφαρμογών **J2EE** σε ένα Διακομιστή εφαρμογών **.NET**.

2.11.4 Ακρωνύμια

Το ακρωνύμιο για την γλώσσα σήμανσης του **ColdFusion** είναι το **CFML**. Όταν αποθηκεύονται στον δίσκο πρότυπα του **ColdFusion**, τους δίνεται συνήθως η κατάληξη «**.cfm**» ή «**.cfml**». Η κατάληξη «**.cfm**» χρησιμοποιείται για τα «εξαρτήματα» του **ColdFusion**. Οι αρχικές καταλήξεις ήταν «**DBM**» ή «**DBML**», οι οποίες αντιστοιχούσαν στο «**Database Markup Language**». Όταν μιλάνε για το **ColdFusion**, οι περισσότεροι χρήστες χρησιμοποιούν το ακρωνύμιο «**CF**», και αυτό χρησιμοποιείται για ένα πλήθος πόρων του **ColdFusion**, όπως οι ομάδες χρηστών (**user groups**) «**CFUGs**» και ιστοσελίδες. Το «**CFMX**» είναι η πιο κοινή συντομογραφία για τις εκδόσεις **6** και **7** (επίσης γνωστές ως **MX**) του **ColdFusion**.

2.12 Αρχιτεκτονική MVC (Model-View-Controller)

2.12.1 Εισαγωγικά

Η αρχιτεκτονική λογισμικού **MVC** (**Model-View-Controller** ή **Μοντέλο-Προβολή-Ελεγκτής** θεωρείται σήμερα ως ένα αρχιτεκτονικό πρότυπο το οποίο χρησιμοποιείται στην μηχανική λογισμικού (**software engineering**). Το πρότυπο απομονώνει την «λογική τομέα» (**domain logic**) (την λογική της εφαρμογής για τον χρήστη) από την είσοδο και την παρουσίαση (**input and presentation**) (δηλαδή την διεπαφή **Graphical User Interface**), επιτρέποντας την ανεξάρτητη ανάπτυξη, δοκιμή και συντήρηση του καθενός.

Το **Μοντέλο** είναι η συγκεκριμένου τομέα αναπαράσταση των δεδομένων επί των οποίων λειτουργεί η εφαρμογή. Η «λογική τομέα» προσθέτει νόημα σε ακατέργαστα δεδομένα (για παράδειγμα, ο υπολογισμός εάν η σημερινή μέρα είναι η μέρα των γενεθλίων, ή τα σύνολα, οι φόροι και τα τέλη αποστολής για προϊόντα αγορών). Όταν ένα **Μοντέλο** αλλά την κατάστασή του, ειδοποιεί τις συσχετισμένες **Προβολές** ώστε να μπορέσουν να ανανεωθούν.

Πολλές εφαρμογές χρησιμοποιούν ένα μόνιμο μηχανισμό αποθήκευσης, όπως είναι μια βάση δεδομένων, για την αποθήκευση δεδομένων. Η αρχιτεκτονική **MVC** δεν αναφέρει συγκεκριμένο στρώμα πρόσβασης δεδομένων (**data access layer**) επειδή υπολογίζεται ότι είναι κάτω από, ή ενθυλακωμένο εντός, του **Μοντέλου**. Τα **Μοντέλα** δεν είναι αντικείμενα πρόσβασης δεδομένων, ωστόσο, σε πολύ απλές εφαρμογές που έχουν μικρή «λογική τομέα» δεν μπορεί να γίνει πραγματική διάκριση. Επίσης, το **ActiveRecord** είναι ένα αποδεκτό πρότυπο σχεδίασης το οποίο συγχωνεύει την «λογική τομέα» και τον κώδικα πρόσβασης δεδομένων.

Η **Προβολή** αναλύει το **Μοντέλο** σε μια μορφή που είναι κατάλληλη για αλληλεπίδραση, δηλαδή συνήθως σε ένα στοιχείο διεπαφής χρήστη. Πολλαπλές **Προβολές** μπορούν να συνυπάρχουν για ένα και μοναδικό **Μοντέλο** για διαφορετικούς σκοπούς. Ο **Ελεγκτής** λαμβάνει της είσοδο και ξεκινάει την δημιουργία μιας απάντησης με την πραγματοποίηση κλήσεων προς τα αντικείμενα του **Μοντέλου.**

Μια εφαρμογή της αρχιτεκτονικής **MVC** μπορεί να αποτελεί μια συλλογή από τριπλέτες **Μοντέλο/Προβολή/Ελεγκτής**, κάθε μία υπεύθυνη για ένα στοιχείο της διεπαφής χρήστη. Η αρχιτεκτονική **MVC** παρατηρείται συχνά σε διαδικτυακές εφαρμογές όπου η **Προβολή** είναι ο κώδικα σε **HTML** ή **XHTML** που παράγεται από την εφαρμογή. Ο **Ελεγκτής** λαμβάνει τις εισόδους **GET** ή **POST** και αποφασίζει το τι θα κάνει με αυτές, και με την σειρά του μεταφέρει την «δουλειά» στα αντικείμενα του τομέα (δηλαδή στο **Μοντέλο**) το οποία περιέχουν τους κανόνες επιχειρήσεων και γνωρίζουν πώς να εκτελέσουν συγκεκριμένα καθήκοντα, όπως είναι η επεξεργασία μια νέας εγγραφής.

2.12.2 Γενική Επισκόπηση

Η αρχιτεκτονική **MVC** περιγράφτηκε για πρώτη φορά το 1979 από τον **Trygve Reenskaug**, ο οποίο εκείνο τον καιρό δούλευε πάνω στην γλώσσα **Smalltalk** στο ερευνητικό κέντρο **Xerox PARC**. Η αρχική υλοποίηση περιγράφεται σε βάθος στην αναφορά του «**Applications Programming in Smalltalk-80: How to use Model-View-Controller**». Έχουν υπάρξει αρκετά παράγωγα της αρχιτεκτονικής **MVC**. Για παράδειγμα, η αρχιτεκτονική **Model View Presenter** χρησιμοποιείται με το «πλαίσιο» **.NET Framework**, και το πρότυπο **XForms** χρησιμοποιεί μια αρχιτεκτονική «**model-view-controller-connector architecture**». Ωστόσο, η βασική αρχιτεκτονική **MVC** εξακολουθεί να είναι δημοφιλής.

Παρόλο που η αρχιτεκτονική **MVC** έρχεται σε διάφορες εκδοχές, η ροή ελέγχου είναι γενικά ως εξής:

1. Ο χρήστης αλληλεπιδρά με την διεπαφή χρήστη με κάποιο τρόπο (για παράδειγμα, πατάει ένα κουμπί).
2. Ο **Ελεγκτής** διαχειρίζεται το συμβάν εισόδου από την διεπαφή χρήστη, συχνά μέσω ενός καταχωρημένου χειριστή (**handler**) ή επανάκλησης (**callback**) και μετατρέπει το συμβάν σε κατάλληλη ενέργεια χρήστη, η οποία είναι κατανοητή από το **Μοντέλο**.
3. Ο **Ελεγκτής** ειδοποιεί το **Μοντέλο** για την ενέργεια του χρήστη, πιθανότατα οδηγώντας σε αλλαγή της κατάστασης του **Μοντέλου**. (Για παράδειγμα, ο **Ελεγκτής** ενημερώνει το καλάθι αγορών του χρήστη.)
4. Μια **Προβολή** υποβάλει ερώτημα στο **Μοντέλο** προκειμένου να παράγει μια κατάλληλη διεπαφή χρήστη (Για παράδειγμα, η **Προβολή** εμφανίζει την λίστα με τα περιεχόμενα του καλαθιού αγορών). Να σημειωθεί ότι η **Προβολή** παίρνει τα δικά της δεδομένα από το **Μοντέλο**. Ο **Ελεγκτής** μπορεί (σε κάποιες υλοποιήσεις) να εκδώσει μια γενική οδηγία προς την **Προβολή** για να πραγματοποιήσει ανάλυση του εαυτού της. Σε άλλες υλοποιήσεις, η **Προβολή** ειδοποιείται αυτόματα από το **Μοντέλο** για αλλαγές στην κατάσταση οι οποίες απαιτούν μια ανανέωση της οθόνης.
5. Η διεπαφή χρήστη περιμένει για περαιτέρω αλληλεπιδράσεις του χρήστη, γεγονός το οποίο ξαναρχίζει τον κύκλο ροής.

Κάποιες υλοποιήσεις όπως η **XForms** της **W3C** χρησιμοποιούν επιπλέον και την έννοια ενός γραφήματος εξάρτησης (**dependency graph**) για να αυτοματοποιήσουν την ανανέωση των **Προβολών**, όταν αλλάζουν τα δεδομένα στο **Μοντέλο**. Ο στόχος της αρχιτεκτονικής **MVC** είναι, με την απόζευξη του **Μοντέλου** και των **Προβολών**, να μειωθεί η πολυπλοκότητα στον αρχιτεκτονικό σχεδιασμό και να αυξηθεί η ευελιξία και η συντηρησιμότητα του κώδικα.

3

Ανάλυση Απαιτήσεων Συστήματος

Σε αυτήν την ενότητα θα περιγραφεί συνοπτικά και γενικά η αρχιτεκτονική της εφαρμογής και θα περιγραφούν με λεπτομέρεια όλες οι λειτουργίες που προσφέρει στον χρήστη.

3.1 Αρχιτεκτονική

Η γενική ιδέα για την δημιουργία της διαδικτυακής εφαρμογής «**Έξυπνες Επαφές**» που είναι το αντικείμενο αυτής της πτυχιακής εργασίας, ήταν η δημιουργία μιας εύρωστης «**μονοσέλιδης**» εφαρμογής (**single-page application**) μέσα σε ένα «πυκνό» πλαίσιο από την πλευρά του Πελάτη. Ένα γενικό περίγραμμα της αρχιτεκτονικής που ακολουθήθηκε για την δημιουργία αυτής της εφαρμογής περιλαμβάνει τα παρακάτω:

- Αρχιτεκτονική «**μονοσέλιδης**» εφαρμογής (**single-page application architecture**)
- Χρήση του **ColdFusion** ως μια **Διαδικτυακή Υπηρεσία**
- Αρχιτεκτονική **MVC** για την **JavaScript**
- Χρήση του **jQuery** ως του «πλαισίου» πυρήνα (**core framework**)

Η τεχνολογία για την πλευρά του Διακομιστή αυτής της εφαρμογής είναι το **ColdFusion**. Παρόλα αυτά, στην πραγματικότητα δεν έγινε πλούσια ενσωμάτωση του **ColdFusion** στην εφαρμογή, απλώς το **ColdFusion** αντιμετωπίζεται ως η διαδικτυακή υπηρεσία της εφαρμογής. Οργανώθηκε ένα πυκνό «πλαίσιο» με χρήση της **JavaScript** στην πλευρά του Πελάτη, και εφαρμόστηκε μια αρχιτεκτονική η οποία ακολουθεί τα πρότυπα της **MVC** αρχιτεκτονικής. Τέλος, η υποδομή αυτής τις εφαρμογής είναι ουσιαστικά το **jQuery**, το οποίο

αποτελεί και το «πλαίσιο» (**framework**) πάνω από το οποίο έχει στηθεί ολόκληρη ο πλευρά του Πελάτη.

Ο τύπος της εφαρμογής είναι σχετικά απλός, είναι μια εφαρμογή διαχείρισης ηλεκτρονικών επαφών, ωστόσο επιλέχτηκε έτσι ώστε να είναι απλή στην κατανόηση, αλλά συνάμα να είναι και αρκετά περίπλοκη στην δομή και την ανάπτυξή της για να παρουσιάσει τις δυνατότητες του **AJAX** που επιλέχθηκαν. Να παρουσιάζει το πώς αυτού του είδους η αρχιτεκτονική μπορεί να απλοποιήσει την ανάπτυξη αντίστοιχων και ίσως μεγαλύτερης εμβέλειας εφαρμογών.

3.2 Περιγραφή Λειτουργιών

Οι λειτουργίες που περιλαμβάνει η εφαρμογή όσον αφορά την διαδραστικότητα με τον χρήστη είναι οι εξής: Αρχικά γίνεται η δημιουργία και ανάρτηση της λίστα των επαφών στην κεντρική σελίδα της εφαρμογής. Στην συνέχεια από την κεντρική σελίδα προσφέρεται η επιλογή της αναζήτησης του ονόματος, της διεύθυνσης ηλεκτρονικού ταχυδρομείου και του τηλεφώνου μιας επαφής, η προβολή των στοιχείων μιας ή περισσοτέρων επαφών που εμφανίζονται κάθε φορά στην λίστα (ανάλογα και με τα αποτελέσματα της αναζήτησης του χρήστη), η επιλογή δημιουργίας μια νέας επαφής, η επιλογή επεξεργασίας μιας ήδη υπάρχουσας επαφής, η διαγραφή μια επαφής, και η δυνατότητα άμεσης επαναφοράς στην κορυφή της σελίδας μέσω του κουμπιού που εμφανίζεται στο κάτω δεξιό μέρος της οθόνης.

3.2.1 Αρχική φόρτωση της εφαρμογής

Με την φόρτωση της εφαρμογής, παρουσιάζεται η κεντρική σελίδα της η οποία αποτελείται από το κύριο μέρος, το μέρος του περιεχομένου, και από το υπόλοιπο κομμάτι που καλύπτεται από την ταπετσαρία της σελίδας. Στο κύριο μέρος εμφανίζονται με την σειρά που περιγράφονται τα παρακάτω. Αρχικά υπάρχει το λογότυπο της σελίδας με ένα εικονίδιο ενός καταλόγου και δίπλα την φράση «**Έξυπνες Επαφές**». Μετά το λογότυπο υπάρχει η μπάρα εργαλείων της εφαρμογής, η οποία περιλαμβάνει μεταξύ άλλων, το πεδίο αναζήτησης και την λέξη «**νέαεπαφή**» που στην πραγματικότητα είναι μια σύνδεση η οποία μεταφέρει τον χρήστη στην φόρμα δημιουργία επαφής. Κάτω από την μπάρα εργαλείων εμφανίζεται η λίστα με τις επαφές, η οποία αποτελείται από πολλά «κουτάκια», ένα για κάθε επαφή. Να σημειωθεί, ότι καθ' όλη την εκτέλεση της εφαρμογής, η εκάστοτε λίστα επαφών που εμφανίζεται (είτε είναι ολόκληρη η λίστα, είτε μια λίστα από φιλτραρισμένες επαφές) είναι με αλφαβητική σειρά, με βάση τα ονόματα των επαφών. Οι επαφές με αγγλικά ονόματα εμφανίζονται πριν από τις επαφές με ελληνικά. Καθώς ο χρήστης διατρέχει την λίστα με τις επαφές κυλώντας προς τα κάτω, εμφανίζεται ένα κουμπί στην κάτω δεξιά γωνία της σελίδας,

το οποίο απεικονίζει ένα βέλος που δείχνει προς τα πάνω. Πατώντας αυτό το κουμπί, ο χρήστης μεταφέρεται στην κορυφή της σελίδας αυτόματα, όχι απευθείας, αλλά με ένα εφέ κύλισης προς τα πάνω.

3.2.2 Πεδίο Αναζήτησης

Το πεδίο αναζήτησης λειτουργεί ουσιαστικά ως ένα πεδίο φιλτραρίσματος των επαφών. Το εκάστοτε γράμμα ή γράμματα που θα πληκτρολογήσει ο χρήστης χρησιμοποιείται ως φίλτρο για να δημιουργηθεί και να εμφανιστεί δυναμικά η ανανεωμένη λίστα με τις επαφές που «πέρασαν» από το φίλτρο. Ο χρήστης μπορεί να πληκτρολογήσει οτιδήποτε επιθυμεί, με ελληνικούς, λατινικούς και αριθμητικούς χαρακτήρες, και το περιεχόμενο του πεδίου θα χρησιμοποιηθεί για το φιλτράρισμα των επαφών. Η διαδικασία φιλτραρίσματος ξεκινάει από τον πρώτο χαρακτήρα που θα πληκτρολογήσει ο χρήστης και συνεχίζει να εφαρμόζεται για όσο χρονικό διάστημα ο χρήστης εισάγει χαρακτήρες στο πεδίο. Για παράδειγμα, εάν ο χρήστης εισάγει τους χαρακτήρες «**bob**» στο πεδίο διαδοχικά, αρχικά θα φιλτραριστούν και θα εμφανιστούν οι επαφές που περιέχουν στα στοιχεία τους τον χαρακτήρα «**b**», στην συνέχεια θα φιλτραριστούν και θα εμφανιστούν αυτές που περιέχουν τους χαρακτήρες «**bo**», και τελικά θα εμφανιστούν οι επαφές που περιέχουν ολόκληρη την λέξη «**bob**». Επισημαίνεται ότι το φιλτράρισμα γίνεται στις επαφές που τα στοιχεία τους **περιέχουν** τους χαρακτήρες που αποτελούν το φίλτρο, και όχι μόνο αυτό απαραίτητα. Δηλαδή, σύμφωνα με το παράδειγμα πιο πάνω, μια επαφή που το όνομά της είναι «**Bobby Dyllan**», θα εμφανιστεί όταν χρησιμοποιηθούν σαν φίλτρο οι χαρακτήρες «**bob**».

Η αναζήτηση των χαρακτήρων γίνεται σε όλα τα στοιχεία των επαφών, δηλαδή και στο όνομα μιας επαφής αλλά ταυτόχρονα και στον αριθμό τηλεφώνου και την διεύθυνση ηλεκτρονικού ταχυδρομείου. Επομένως, σύμφωνα με το προηγούμενο παράδειγμα, στο τέλος του φιλτραρίσματος θα εμφανίζονται μόνο οι επαφές που μέσα στο όνομα ή την διεύθυνση ηλεκτρονικού ταχυδρομείου τους, περιέχονται οι χαρακτήρες «**bob**» με την σειρά που πληκτρολογήθηκαν (δηλαδή **δεν** θα εμφανίζονται οι επαφές που π.χ. το όνομά τους περιέχει τους χαρακτήρες «**obb**»).

Ενώ ο χρήστης βρίσκεται εντός του πεδίου αναζήτησης έχει δύο επιλογές-δυνατότητες, οι οποίες εκτελούνται με το πάτημα πλήκτρων του πληκτρολογίου. Συγκεκριμένα, ο χρήστης έχει την δυνατότητα να πατήσει το κουμπί «**Enter**» του πληκτρολογίου, με αποτέλεσμα οι πρώτες τρεις επαφές τις λίστας θα αναδιπλωθούν προς τα κάτω, με ένα εφέ κίνησης, έτσι ώστε να εμφανιστούν τα επιπλέον στοιχεία τους (δηλαδή ο αριθμός τηλεφώνου και η διεύθυνση ηλεκτρονικού ταχυδρομείου). Η αναδίπλωση των τριών πρώτων επαφών, και όχι περισσοτέρων, επιλέχθηκε για λόγους κυρίως επιδόσεων, για να μην υπάρχουν υπερβολικά πολλά εφέ κίνησης που θα εκτελούνται ταυτόχρονα, και για λόγους χρηστικότητας, αφού

βάση περιεχομένου, μετά την τρίτη επαφή που αναδιπλώνεται, δεν απομένει χώρος για να εμφανιστεί καμία επιπλέον επαφή εντός του ορατού πεδίου της σελίδας. Η δεύτερη δυνατότητα που έχει ο χρήστης είναι να πατήσει το συνδυασμό των πλήκτρων «**Shift+Enter**» του πληκτρολογίου του, γεγονός το οποίο θα φέρει ως αποτέλεσμα την μεταφορά του στην φόρμα επεξεργασίας της πρώτης επαφής που βρίσκεται την δεδομένη στιγμή στην λίστα επαφών του (δηλαδή εάν μετά από μια αναζήτηση ο χρήστης έχει μπροστά του δύο επαφές, θα μεταφερθεί στην φόρμα επεξεργασίας της πρώτης από την λίστα).

Εάν ο χρήστης δεν έχει πληκτρολογήσει κάτι στο πεδίο αναζήτησης, ή εάν αυτό που έχει πληκτρολογήσει δεν έχει εμφανίσει κανένα αποτέλεσμα στην λίστα των επαφών, με το πάτημα του συνδυασμού κουμπιών «**Shift+Enter**» ή με το κλικ του ποντικιού του πάνω στην σύνδεση «**νέαεπαφή**», θα μεταφερθεί στην φόρμα δημιουργίας νέας επαφής, η οποία εάν έχει πληκτρολογηθεί προηγουμένως κάτι στο πεδίο αναζήτησης, θα γεμίσει το πεδίο του ονόματος της νέας επαφής με το περιεχόμενο που είχε πληκτρολογηθεί. Για παράδειγμα, εάν ο χρήστης πληκτρολογήσει τους χαρακτήρες «**Ουρανία**» και δεν υπάρχουν αποτελέσματα, πατώντας «**Shift+Enter**» θα μεταφερθεί στην φόρμα δημιουργίας νέας επαφής, όπου το πεδίο του ονόματος της νέας επαφής θα περιέχει ήδη το όνομα «**Ουρανία**» και επίσης θα είναι το τρέχον κεντραρισμένο πεδίο για να δεχτεί την περαιτέρω είσοδο του χρήστη.

Εάν ο χρήστης βρίσκεται στο πεδίο αναζήτησης και πατήσει τον συνδυασμό των κουμπιών «**ALT+Enter**» από το πληκτρολόγιό του, θα ξεκινήσει η λειτουργία διαγραφής της πρώτης επαφής που εμφανίζεται εκείνη την δεδομένη στιγμή στην λίστα. Αρχικά ο χρήστης θα ειδοποιηθεί με ένα παράθυρο ειδοποίησης από την εφαρμογή το οποίο θα του παρέχει τις δύο επιλογές, «**ΟΚ**» και «**Ακύρωση**». Στην συνέχεια, και εφόσον επιλέξει να επικυρώσει την επιλογή του για διαγραφή της επαφής, η διαγραφή θα εκτελεστεί και η λίστα των επαφών θα ενημερωθεί και θα ανανεωθεί σύμφωνα με τα καινούργια δεδομένα.

3.2.3 Λίστα Επαφών

Η λίστα με τις επαφές δημιουργείται, φιλτράρεται και ανανεώνεται δυναμικά, σύμφωνα με τις ενέργειες του χρήστη. Εφόσον υπάρχουν επαφές στην λίστα, εμφανίζονται οργανωμένες αλφαβητικά σε διαδοχικά μακρόστενα «κουτιά». Κάθε «κουτί» περιέχει από αριστερά προς τα δεξιά τα εξής: Το όνομα της επαφής, και τρείς λέξεις-συνδέσεις, τις «**περισσ.**», «**επεξ.**» και «**διαγραφή**». Η λέξη-σύνδεση «**περισσ.**» είναι σύντμηση της λέξης «**περισσότερα**» και εφόσον πατηθεί εμφανίζονται, με ένα εφέ αναδίπλωσης προς τα κάτω, τα στοιχεία της εκάστοτε επαφής, τα οποία αποτελούνται από το όνομα της επαφής, τον αριθμό τηλεφώνου και την διεύθυνση ηλεκτρονικού ταχυδρομείου. Ο χρήστης μπορεί να αναδιπλώσει όσα κουτιά επιθυμεί ταυτόχρονα, για να εμφανίσει τα στοιχεία όσων επαφών επιθυμεί.

Δίπλα στην σύνδεση «**περισσ.**», υπάρχει η σύνδεση «**επεξ.**» η οποία προέρχεται από την σύντμηση της λέξης «**επεξεργασία**». Πατώντας αυτήν την σύνδεση μια επαφής, ο χρήστης μεταφέρεται στην φόρμα επεξεργασίας επαφής, όπου μπορεί να επεξεργαστεί τα στοιχεία της επαφής αυτής. Τέλος, η τελευταία επιλογή που εμφανίζεται στο «κουτί» μιας επαφής, είναι η σύνδεση «**διαγραφή**». Όπως δηλώνει και το όνομά της, η σύνδεση αυτή διαγράφει την αντίστοιχη επαφή. Πριν την οριστική διαγραφή, ο χρήστης ενημερώνεται με ένα παράθυρο επιβεβαίωσης, στο οποίο ερωτάται εάν είναι σύμφωνος για την ενέργειά που πρόκειται να εκτελεστεί και του προσφέρονται οι επιλογές «**ΟΚ**» και «**Άκυρο**», οι οποίες επιβεβαιώνουν ή ακυρώνουν την διαγραφή αντίστοιχα. Με την διαγραφή μιας επαφής, η λίστα ενημερώνεται και επαναφορτώνεται.

3.2.4 Φόρμα Επεξεργασίας και Δημιουργία Νέας Επαφής

Οι προαναφερθείσες φόρμες, επεξεργασίας και δημιουργία νέας επαφής, είναι ουσιαστικά η ίδια φόρμα, η οποία στην περίπτωση της επεξεργασίας μια επαφής εμφανίζεται με τα πεδία της να περιέχουν τα στοιχεία της επαφής, ενώ στην δεύτερη περίπτωση εμφανίζονται κενά (στην δεύτερη περίπτωση μπορεί να υπάρχει περιεχόμενο στο πεδίο ονόματος μιας επαφής, ανάλογα με την ενέργεια που οδήγησε την εφαρμογή στην φόρμα, όπως περιγράφεται παραπάνω). Τα πεδία της φόρμας είναι τα αντίστοιχα με τα στοιχεία μια εφαρμογής, δηλαδή «**Όνομα**», «**Τηλέφωνο**» και «**Email**». Από τα πεδία της φόρμας, μόνο το πεδίο «**Όνομα**» είναι υποχρεωτικό, ενώ τα άλλα μπορούν να παραμείνουν κενά. Στην περίπτωση που ο χρήστης προσπαθήσει να αποθηκεύσει μια επαφή, χωρίς να έχει δώσει όνομα για αυτήν, η εφαρμογή εμφανίζει ένα μήνυμα ενσωματωμένο πάνω ακριβώς από το πεδίο του ονόματος, το οποίο ειδοποιεί τον χρήστη να εισάγει ένα όνομα. Το μήνυμα που εμφανίζεται αναγράφει: «**Παρακαλώ εισάγετε ένα όνομα**».

Στο πάνω αριστερό μέρος της φόρμας, υπάρχει η σύνδεση με το όνομα «**πίσω**», το οποίο μεταφέρει τον χρήστη πίσω στην κεντρική σελίδα της εφαρμογής, την σελίδα δηλαδή με την λίστα επαφών, χωρίς να αποθηκεύεται καμία αλλαγή και χωρίς να εμφανίζεται καμία ειδοποίηση στον χρήστη. Το κουμπί αυτό έχει δημιουργηθεί για γρήγορη επιστροφή στην λίστα επαφών.

Στο κάτω μέρος της φόρμας, υπάρχουν τα κουμπιά «**Αποθήκευση & Κλείσιμο**» και «**Ακύρωση**» με την αναγραφόμενη σειρά. Πατώντας το κουμπί «**Αποθήκευση & Κλείσιμο**» η εφαρμογή αποθηκεύει ότι αλλαγές έχουν γίνει από τον χρήστη, συμπεριλαμβανομένης και της δημιουργίας μιας νέας επαφής εάν αυτό είναι απαραίτητο, και στην συνέχεια μεταφέρει τον χρήστη πίσω στην κεντρική σελίδα. Με το πάτημα του κουμπιού θα γίνει και ο έλεγχος για το εάν το πεδίο του ονόματος είναι κενό, και στην συνέχεια θα εμφανιστεί το μήνυμα που περιγράφεται παραπάνω. Το κουμπί «**Ακύρωση**», που ουσιαστικά είναι κουμπί-σύνδεση, με

το που πατηθεί θα εμφανίσει στον χρήστη ένα παράθυρο επιβεβαίωσης, το οποίο ειδοποιεί τον χρήστη ότι πρόκειται να χάσει τις αλλαγές που έχει κάνει, και εάν ο χρήστης επιβεβαιώσει την εντολή του, θα ακυρώσει τις αλλαγές και θα μεταφέρει τον χρήστη πίσω στην κεντρική σελίδα.

3.2.5 Η συμπεριφορά της διεύθυνσης URL της εφαρμογής

Εάν παρατηρήσει κανείς την διεύθυνση **URL** της εφαρμογής που εμφανίζεται στο πρόγραμμα περιήγησης, θα διαπιστώσει ότι πέραν του κομματιού που παραμένει σταθερό, υπάρχει και ένα κομμάτι, μετά τον χαρακτήρα «#» οποίο αλλάζει σύμφωνα με την δεδομένη κατάσταση που βρίσκεται η εφαρμογή. Έτσι π.χ. όταν βρισκόμαστε στην φόρμα επεξεργασίας της επαφής με εσωτερικό αναγνωριστικό (**ID**) το νούμερο 67, στην διεύθυνση **URL** πέραν του σταθερού μέρους βλέπουμε να αναγράφεται το: «**#/contacts/edit/67**», ενώ βρισκόμενη στην φόρμα δημιουργίας νέας επαφής, βλέπουμε να αναγράφεται το: «**#/contacts/add**». Αυτές οι αλλαγές παράγονται δυναμικά και ανανεώνονται σε κάθε αλλαγή της κατάστασης της εφαρμογής, έτσι ώστε το πρόγραμμα περιήγησης να κρατάει στο ιστορικό του της διάφορες καταστάσεις, και έτσι να λειτουργούν οι επιλογές «**Πίσω**» και «**Εμπρός**» που προσφέρει στον χρήστη. Επίσης με αυτόν τον τρόπο ο χρήστης μπορεί να αποθηκεύσει μια δεδομένη κατάσταση της εφαρμογής στους σελιδοδείκτες του. Παρόλα αυτά το κύριο τμήμα της διεύθυνσης **URL** της εφαρμογής παραμένει σταθερό και δεν αλλάζει με τέτοιο τρόπο ώστε να χρειάζεται επαναφόρτωση ή αλλαγή κάποιας σελίδας.

Εδώ ολοκληρώνονται οι λειτουργίες-δυνατότητες που προσφέρει στον χρήστη η εφαρμογή.

4

Σχεδίαση Συστήματος

Σε αυτήν την ενότητα θα περιγραφεί η σχεδίαση της εφαρμογής. Αναλύεται η γενική αρχιτεκτονική της εφαρμογής και στην συνέχεια αναλύονται οι επιμέρους αρχιτεκτονικές των δυο πλευρών της εφαρμογής, του Πελάτη και του Διακομιστή.

4.1 Αρχιτεκτονική Εφαρμογής

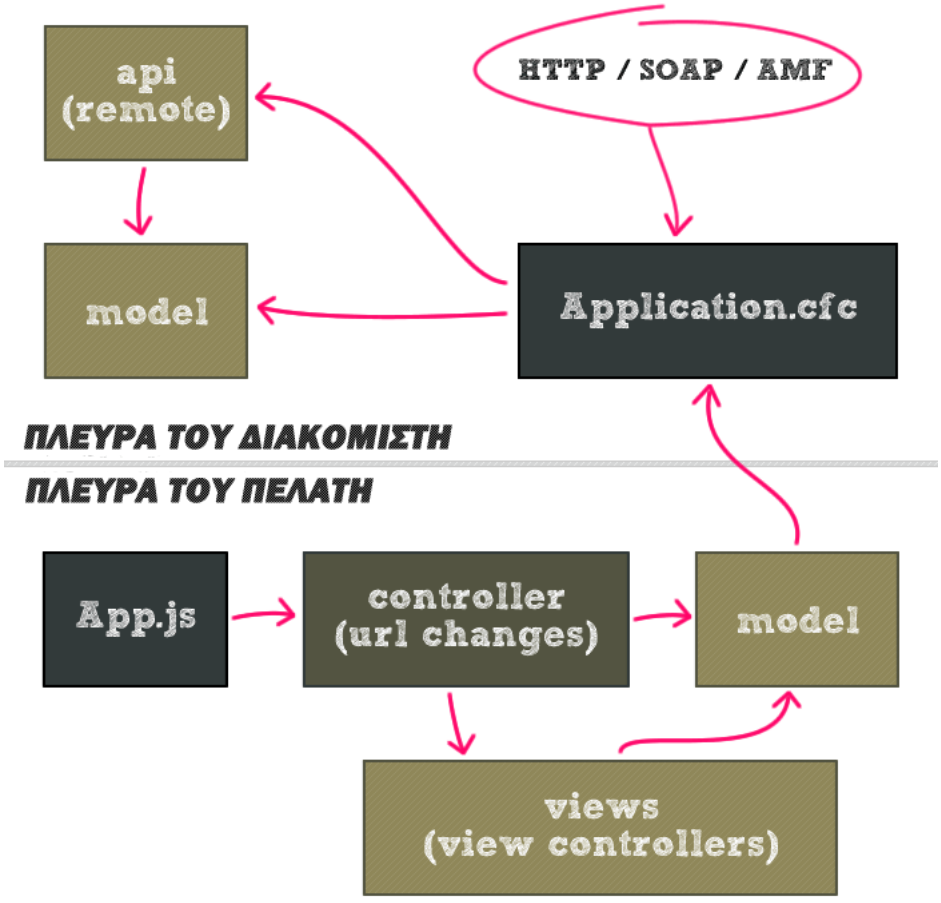
Η αρχιτεκτονική της εφαρμογής περιλαμβάνει τις δυο πλευρές δράσης της εφαρμογής, την πλευρά του Πελάτη και την πλευρά του Διακομιστή. Μια γενική μορφή της αρχιτεκτονικής για όλο το πεδίο αναφοράς όλης της εφαρμογής, φαίνεται στο **Σχήμα 5.1**.

Όσον αφορά την πλευρά του Διακομιστή, όλες οι αιτήσεις περνάνε από το αρχείο «**Application.cfc**». Αυτές μπορεί να είναι αιτήσεις **SOAP**, αιτήσεις **HTTP**, ή αιτήσεις **AMF** (**Action Message Format**). Το «**Application.cfc**» είναι το κύριο «τούνελ» που οδηγεί στην εφαρμογή **ColdFusion** στην πλευρά του Διακομιστή. Το «**Application.cfc**», εφόσον δεχτεί κάποιο είδος αίτησης, στην συνέχεια επικοινωνεί με το **Μοντέλο** (**model**) ή με τις απομακρυσμένες διεπαφές **API** (**remote API**), οι οποίες με την σειρά τους μπορούν επίσης να επικοινωνήσουν με το **Μοντέλο.**

Όσον αφορά την πλευρά του Πελάτη, το κύριο «τούνελ» για την επικοινωνία αυτή τη φορά είναι το αρχείο «**application.js**». Είναι το αντίστοιχο του «**Application.cfc**» αλλά για την πλευρά του Πελάτη. Το «**application.js**» είναι το «πλαίσιο» της **JavaScript**, το **jQuery** «πλαίσιο» (**jQuery framework**), το οποίο είναι ο πυρήνας της εφαρμογής, όσον αφορά την

πλευρά του Πελάτη. Αναλαμβάνει την επικοινωνία με τον **Ελεγκτή** (**controller**) όταν αντιλαμβάνεται αλλαγές στην διεύθυνση του **URL**, οποίος με την σειρά του αποφασίζει εάν θα επικοινωνήσει με το **Μοντέλο** (**model**) ή με τις **Προβολές** (**Views**). Οι **Προβολές** με την σειρά τους, που για καλύτερη κατανόηση μπορούν να θεωρηθούν και ως **Ελεγκτές Προβολής** (**View Controllers**), μπορούν επίσης να επικοινωνήσουν με το **Μοντέλο**.

Αυτό που αξίζει να αναφερθεί εδώ, είναι πως όσον αφορά την επικοινωνία μεταξύ των δυο πλευρών, το μόνο μέρος της πλευράς του Πελάτη που «μιλάει» με τον Διακομιστή είναι το **Μοντέλο**. Το «**application.js**», ο **Ελεγκτής** και ασφαλώς οι **Προβολές**, δεν γνωρίζουν ουσιαστικά την ύπαρξη ενός Διακομιστή, καθώς δεν έχουν άμεση επικοινωνία με κάποιον. Επομένως, οποιαδήποτε μορφή πληροφορίας που θα χρειαστεί να ληφθεί από τον Διακομιστή, διεξάγεται στο επίπεδο του **Μοντέλου**.



Σχήμα 5.1 Διάγραμμα της αρχιτεκτονικής της εφαρμογής

4.2 Πλευρά του Πελάτη

Ο πλευρά του Πελάτη αποτελεί το βασικό μέρος της εφαρμογής, και βασίζεται κατά κύριο λόγο στην τεχνολογία και τις ευκολίες που προσφέρει η βιβλιοθήκη **jQuery** της **JavaScript**. Επειδή η εφαρμογή έχει σχεδιαστεί στα πλαίσια μιας «**μονοσέλιδης**» εφαρμογής (**single-**

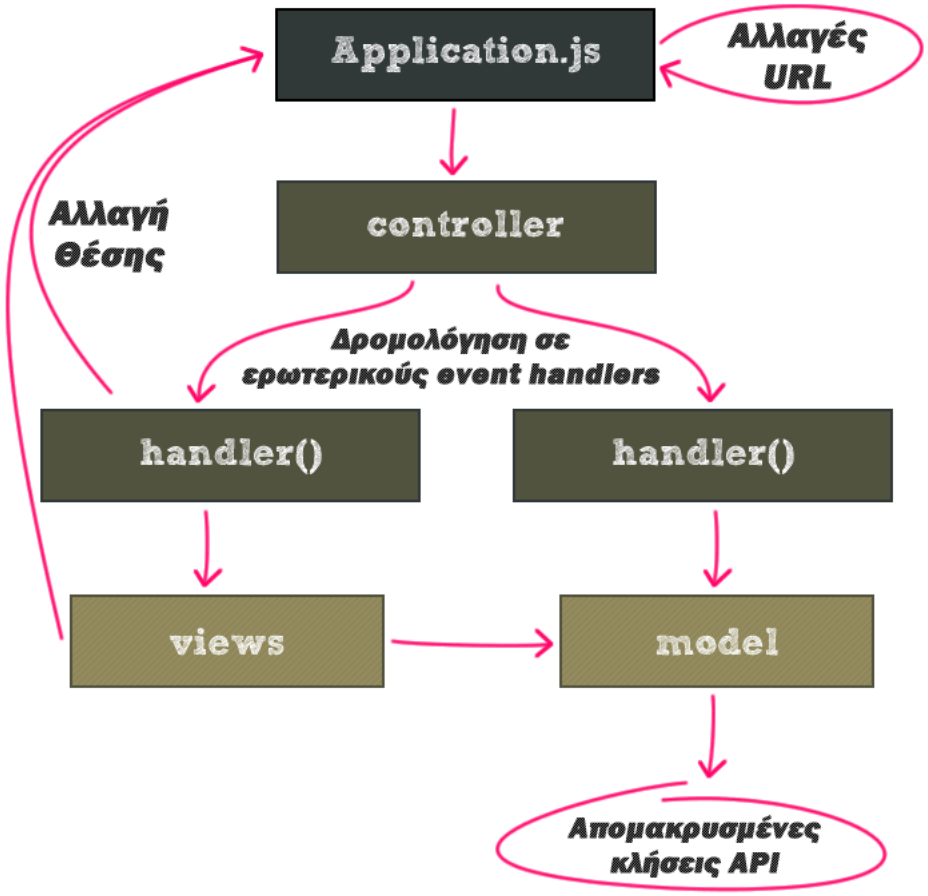
application), η αρχική σελίδα (**Index page**) ουσιαστικά **είναι** και η εφαρμογή. Το αρχείο «**application.js**» είναι το «πλαίσιο» (**framework**), δηλαδή ο κύριος σκελετός, για την εφαρμογή όσον αφορά την πλευρά του Πελάτη, όπως αντίστοιχα, το αρχείο «**Application.cfc**» είναι ο κύριος σκελετός της εφαρμογής στην πλευρά του Διακομιστή.

Η εφαρμογή παρακολουθεί την διεύθυνση του **URL** για αλλαγές στο μεταβλητό μέρος του δηλαδή το μέρος που, όπως αναλύθηκε και στην προηγούμενη ενότητα, ξεκινάει μετά το σύμβολο «#» (το λεγόμενο και **hash** σύμβολο). Αυτές οι αλλαγές λέγονται και «**αλλαγές-hash**» («**hash changes**») από το όνομα του συμβόλου ή «**αλλαγές αποσπάσματος**» (**fragment changes**) καθώς αυτό που αλλάζει είναι ένα συγκεκριμένο απόσπασμα από την διεύθυνση **URL**. Όταν η εφαρμογή ανιχνεύει «αλλαγές-hash» όπως είναι η αλλαγή από την αρχική κατάσταση (που είναι «**#/contacts/**») στην κατάσταση επεξεργασίας της επαφής με το αναγνωριστικό 123 (που είναι «**#/contacts/edit/123**») γνωρίζει πως πρέπει να «μιλήσει» με τον **Ελεγκτή** και να τον ειδοποιήσει ότι έγινε μια «αλλαγή-hash». Ο **Ελεγκτής** με την σειρά του, γνωρίζει πώς να αντιστοιχίζει αυτές τις «αλλαγές-hash» σε εσωτερικούς «διαχειριστές συμβάντος» («**event handlers**»), οι οποίοι με την σειρά τους μπορούν να αναθέσουν τον έλεγχο είτε στο **Μοντέλο** είτε στις **Προβολές**.

Η παραπάνω αρχιτεκτονική μπορεί να αποδοθεί και σχεδιαγραμματικά, όπως φαίνεται στο **Σχήμα 5.2**, για καλύτερη κατανόηση. Όπως φαίνεται και στο σχήμα έχουμε το «**application.js**», το οποίο είναι ο πυρήνας της πλευράς του Πελάτη, που παρακολουθεί την διεύθυνση **URL** για αλλαγές. Όταν ανιχνεύσει αλλαγές, «μιλάει» με τον **Ελεγκτή**, και τον ειδοποιεί για αυτές τις αλλαγές. Με την σειρά του ο **Ελεγκτής** δρομολογεί αυτές τις «αλλαγές-hash» στους εσωτερικούς διαχειριστές συμβάντος, οι οποίοι μπορούν να «μιλήσουν» είτε με το **Μοντέλο**, είτε με τις **Προβολές**. Οι ίδιες οι **Προβολές** μπορούν να «μιλήσουν» με το **Μοντέλο** αυτόνομα, για να συλλέξουν πληροφορίες. Το μόνο μέρος της εφαρμογής στην πλευρά του Πελάτη που πρέπει να επικοινωνήσει με τον Διακομιστή, είναι το **Μοντέλο**, το οποίο θα επικοινωνεί με την χρήση απομακρυσμένων κλήσεων **API** (**Remote API calls**). Με αυτόν τον τρόπο, η πλευρά του Διακομιστή αντιμετωπίζεται ως μια διαδικτυακή υπηρεσία, και όχι σαν κάτι που ενσωματώνεται με την μεθοδολογία «αίτησης-απάντησης».

Καθώς ο **Ελεγκτής** και οι **Προβολές** ανταποκρίνονται σε συμβάντα που λαμβάνουν χώρα εντός του συστήματος, μπορούν επίσης να ζητήσουν από την εφαρμογή να αλλάξει θέσεις (δηλαδή διευθύνσεις). Αυτό είναι κατά κάποιο τρόπο το αντίστοιχο κομμάτι στον Διακομιστή, με την κλήση της ετικέτας «**CFLocation**», όπου, για παράδειγμα, ο **Ελεγκτής** λαμβάνει μια εντολή διαγραφής, εκτελεί την διαγραφή και στην συνέχεια πρέπει να ειδοποιήσει την εφαρμογή ότι εκτέλεσε την εντολή και ότι θα πρέπει να «γυρίσει» (η εφαρμογή) τον χρήστη πίσω στη λίστα των επαφών. Γυρίζοντας πίσω στο σχεδιάγραμμά μας,

μόλις θα ζητηθεί από την εφαρμογή να αλλάξει θέση, θα ενεργοποιηθεί η ίδια ροή εργασίας (**Work Flow**), με την διεύθυνση **URL** να αλλάζει θέση και να «πηγαίνει» στην λίστα των επαφών και να ξεκινάει η ίδια διαδικασία πάλι από την αρχή.

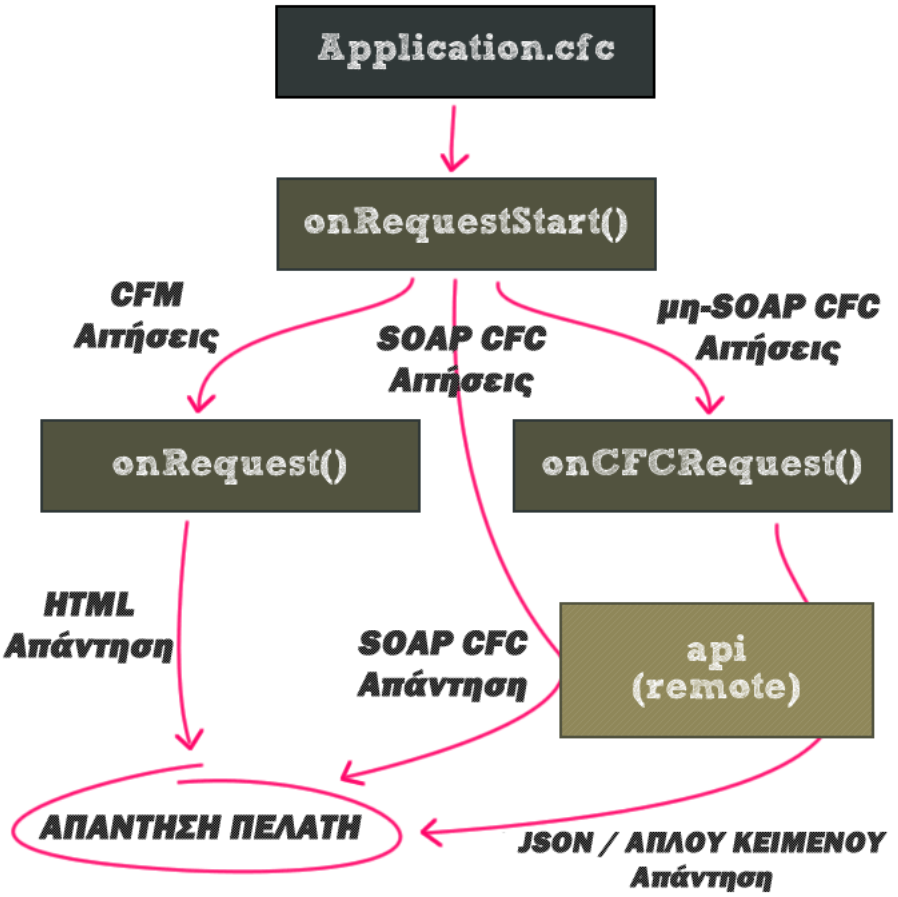


Σχήμα 5.2 Διάγραμμα της αρχιτεκτονικής στην πλευρά του Πελάτη

4.3 Πλευρά του Διακομιστή

Η πλευρά του Διακομιστή αποτελεί την διαδικτυακή υπηρεσία της εφαρμογής, και βασίζεται κατά κύριο λόγο στην τεχνολογία **ColdFusion**. Οποιαδήποτε επικοινωνία με την πλευρά του Διακομιστή «περνάει» μέσω του αρχείου «**Application.cfc**». Ανεξάρτητα από τον τύπο των αιτήσεων που δέχεται ο Διακομιστής, είτε είναι **CFM Αιτήσεις**, είτε είναι απομακρυσμένες **CFC Αιτήσεις**, όλες «περνάνε» μέσω του αρχείου «**Application.cfc**». Είναι το εσωτερικό, το έμφυτο, «πλαίσιο» σκελετός της εφαρμογής όσον αφορά το κομμάτι του **ColdFusion**. Η πληροφορία που επιστρέφεται στον Πελάτη, μετά από το πέρασμα μέσα από το «**Application.cfc**», μπορεί να είναι πλήρως μεταβλητή. Μπορεί να είναι σε μορφή **HTML**, μπορεί να είναι σε μορφή **JSON**, σε μορφή κειμένου (**text**), σε μορφή **SOAP**, μπορεί ακόμη και να είναι σε μορφή **AMF** ή προσαρμοσμένης **XML** (**customized XML**) και σε πολλές

άλλες μορφές. Αυτή είναι μια από τις πολλές επιλογές-δυνατότητες που προσφέρει η τεχνολογία **ColdFusion**.



Σχήμα 5.3 Διάγραμμα της αρχιτεκτονικής στην πλευρά του Διακομιστή

Στο **Σχήμα 5.3** φαίνεται ένα γενικό σχεδιάγραμμα της αρχιτεκτονικής και της ροής εργασίας της πλευράς του Διακομιστή. Όπως φαίνεται στο σχήμα, το αρχείο «**Application.cfc**» είναι στην κορυφή της αρχιτεκτονικής πυραμίδας. Εσωτερικά του «**Application.cfc**», καλείται ο διαχειριστής συμβάντων (**event handler**) «**onRequestStart()**», ο οποίος καλείται «σιωπηρά» (δηλαδή αυτόματα και στο παρασκήνιο, implicitly) από τον «**ColdFusion Server**» χωρίς να χρειαστεί επιπλέον προγραμματισμός. Ο «**onRequestStart()**» επιμελείται των αποφάσεων, όσον αφορά την δρομολόγηση της εισερχόμενης αίτησης. Εάν είναι απλά μια **CFM Αίτηση**, θα την δρομολογήσει στον διαχειριστή συμβάντων «**onRequest()**», ο οποίος αναλαμβάνει την συμπερίληψη (**including**) των **CFM** σελίδων. Εάν είναι μια **μη-SOAP CFC Αίτηση**, θα την δρομολογήσει στον διαχειριστή συμβάντων «**onCFCRequest()**», ο οποίος με την σειρά του θα την δρομολογήσει στην κατάλληλη απομακρυσμένη διεπαφή **API** (**Remote API**). Εάν είναι μια **SOAP CFC Αίτηση**, επειδή το **ColdFusion** λειτουργεί πολύ καλά με το **SOAP** και μάλιστα στο παρασκήνιο, όπως για παράδειγμα στην μετατροπή των πακέτων **SOAP** σε «κατασκευάσματα» του **ColdFusion**, δεν θέλουμε να παρέμβουμε στο έργο του καθώς

εκτελεί ένα μεγάλο μέρος του φόρτου εργασίας αυτόματα για μας, επομένως δεν θα μπούμε στο δρόμο του και θα το αφήσουμε να αναλάβει τις **SOAP** αιτήσεις από μόνο του. Με αυτήν την λογική, μια **SOAP CFC Αίτηση** δεν θα δρομολογηθεί σε κάποιον από τους δυο προαναφερθέντες διαχειριστές συμβάντων αλλά θα «μιλάει» απευθείας στις απομακρυσμένες διεπαφές **API**, με την βοήθεια πάντα του **ColdFusion**.

Ανεξάρτητα από την πορεία δρομολόγησης της αίτησης, όλη η πληροφορία που παράγεται κατά την απάντηση θα επιστραφεί στον Πελάτη. Επομένως, ανεξάρτητη από τον τύπο της αίτησης και κατά συνέπεια και της απάντησης, είτε είναι μια **HTML** απάντηση, είτε είναι μία **SOAP CFC Απάντηση**, είτε είναι μια απάντηση δεδομένων σε μορφή όπως **JSON** ή απλού κειμένου, όλη η πληροφορία της απάντησης θα «εισρεύσει» προς τον Πελάτη.

5

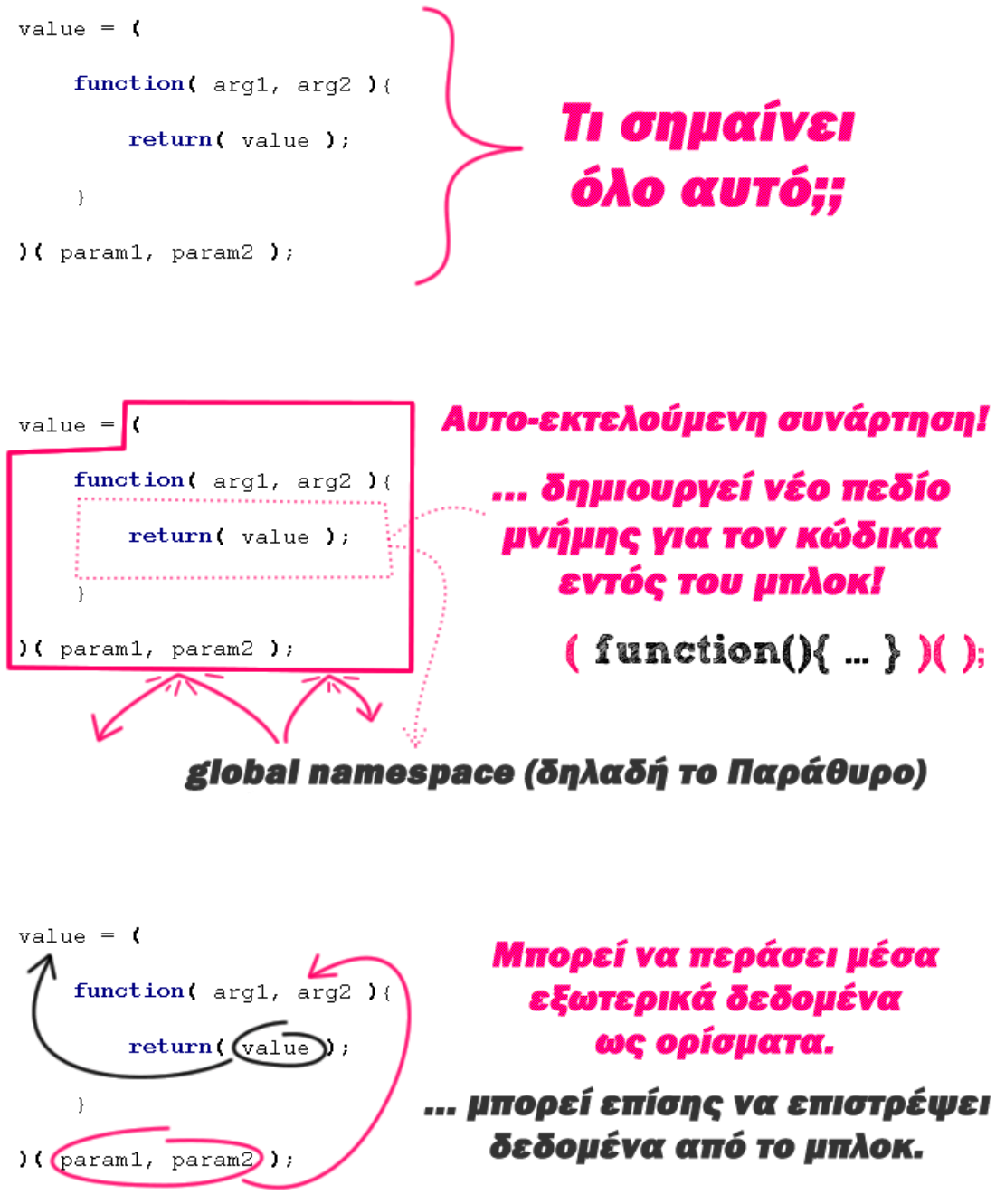
Υλοποίηση

Σε αυτήν την ενότητα θα αναλυθούν λεπτομερώς θέματα υλοποίησης του συστήματος. Τα θέματα αυτά αφορούν τεχνικές που ακολουθήθηκαν και αξίζουν προσοχής καθώς και κομμάτια κώδικα που παρουσιάζουν ενδιαφέρον και άλλα τεχνικά θέματα.

5.1 Πλευρά του Πελάτη

5.1.1 Αυτό-εκτελούμενα μπλοκ συναρτήσεων

Πριν προχωρήσουμε περαιτέρω στα θέματα υλοποίησης του κώδικα της εφαρμογής, θα πρέπει να αναλυθούν κάποιες τεχνικές της **JavaScript** οι οποίες χρησιμοποιούνται ευρέως σε όλο τον προγραμματισμό αυτής της εφαρμογής. Στο **Σχήμα 6.1** υπάρχει στην κορυφή ένα μπλοκ κώδικα. Μέσα του περιέχεται μια συνάρτηση, ενδιάμεσα των παρενθέσεων, και ακολουθούν οι επόμενες παρενθέσεις («(**param1**, **param2**)») οι οποίες ουσιαστικά αποτελούν την κλήση της συνάρτησης. Αυτή η δυνατότητα που προσφέρει η **JavaScript** αποτελεί μια εκπληκτική τεχνική η οποία προσφέρει πολλά πλεονεκτήματα αλλά ίσως είναι κάπως περίπλοκη στην κατανόηση. Αυτό που ουσιαστικά γίνεται, είναι η δημιουργία μια αυτό-εκτελούμενης συνάρτησης. Έτσι έχουμε μια ανώνυμη συνάρτηση, την οποία με το να την περικλείουμε σε παρενθέσεις και να προσθέτουμε τις ακόλουθες παρενθέσεις, θα οδηγήσει στην άμεση εκτέλεση της συνάρτησης.



Σχήμα 6.1 Η τεχνική της αυτο-εκτελούμενης συνάρτησης στην JavaScript

Με αυτόν τρόπο, δημιουργείτε μια «φούσκα» εκτέλεσης, η οποία εκτελείται εντός ενός νέου, ειδικά αφιερωμένου σ' αυτήν, πεδίου μνήμης, για όλον τον κώδικα που εκτελείται εσωτερικά αυτής της συνάρτησης. Αυτό είναι ένα πολύ ισχυρό χαρακτηριστικό, γιατί ουσιαστικά αυτό που κάνει είναι να δημιουργεί μια «ασπίδα» για την συνάρτηση, η οποία την διαχωρίζει από τον ενιαίο χώρο των μεταβλητών στην μνήμη (**global namespace**) δηλαδή το «**Παράθυρο**» της εφαρμογής. Το «**global namespace**» δεν μπορεί να μπει εντός του πεδίου που έχει δημιουργηθεί για αυτήν την «φούσκα» εκτέλεσης. Το πλεονέκτημα αυτής της τεχνικής είναι ότι από τα εσωτερικό της «φούσκας» μπορούμε να επικοινωνήσουμε με το «**global namespace**» εάν χρειαστεί. Αυτή η τεχνική μας επιτρέπει να έχουμε κομμάτια κώδικα που

δεν «γεμίζουν» το «**global namespace**» με τυχαίο κώδικα, αλλά που έχουν την δυνατότητα να επηρεάζουν το «**global namespace**» εάν αυτό είναι απαραίτητο.

Κατά την εκτέλεση αυτής της συνάρτησης, μπορούν να εκτελεστούν οι κανονικές λειτουργίες μια συνάρτησης. Μπορούμε να «περάσουμε» πληροφορία με την μορφή ορισμάτων (**param1**, **param2**) μέσα στην συνάρτηση, όπως επίσης μπορούμε να «επιστρέψουμε» πληροφορία από την συνάρτηση. Όπως φαίνεται και στο τελευταίο σκέλος του **Σχήματος 6.1**, τα ορίσματα «**param1**» και «**param2**» περνάνε μέσα στην συνάρτηση ως «**arg1**» και «**arg2**». Επίσης, επιστρέφεται η τιμή «**value**» η οποία «περνάει» στην μεταβλητή «**value**» που βρίσκεται έξω από αυτό το μπλοκ κώδικα.

Στην συνέχεια αυτής της ενότητας, και καθώς θα αναλύεται συγκεκριμένος κώδικας της εφαρμογής, θα υπάρξουν παραδείγματα και αναφορές σε αυτήν την τεχνική, για καλύτερη κατανόηση της.

5.1.2 Το «application.js» Είναι ένα Εργοστάσιο Αντικείμενων

Το αρχείο «**application.js**», το οποίο είναι ο σκελετός «πλαισίου» (**core framework**) της εφαρμογής στην πλευρά του Πελάτη, είναι ένα «**εργοστάσιο αντικειμένων**» (**object factory**). Εξηγώντας τον όρο αυτό, το αρχείο «**application.js**», είναι υπεύθυνο για την διατήρηση μιας συλλογής από «ορισμούς» κλάσεων (class definitions) και «**παρουσίες**» (**instances**) από «**singletons**» (Το «**singleton**» είναι ένα σχεδιαστικό πρότυπο, το οποίο χρησιμοποιείται για τον περιορισμό της «συγκεκριμενοποίησης» (**instantiation**) μιας κλάσης σε ένα αντικείμενο). Είναι επίσης υπεύθυνο για την δυνατότητα προσθήκης και ανάκτησης των στοιχείων αυτής της συλλογής. Με λίγα λόγια το «**application.js**» διαχειρίζεται την «συγκεκριμενοποίηση» των κλάσεων και την αποθήκευση των «**singletons**».

Φυσικά, το «**application.js**» δεν μπορεί μαγικά να τα κάνει όλα αυτά, θα πρέπει λοιπόν να του δηλώσουμε τις κλάσεις που χρησιμοποιούμε στην εφαρμογή. Ο τρόπος που χρησιμοποιείται για την δήλωση αυτών των κλάσεων, είναι με την χρήση των τριών ακόλουθων μεθόδων:

- application.**addController**(instance);
- application.**addModel**(instance | class);
- application.**addView**(instance | class);

Όπως είναι φανερό, και οι τρεις μέθοδοι καλούνται από το «**application**». Όταν το «**application.js**» ξεκινάει, δημιουργεί μια νέα «παρουσία» (**instance**) του δικού του «πλαισίου» (**framework**) και την αποθηκεύσει στο «**window.application**». Εφόσον το «window» είναι το «**global namespace**» μπορούμε να αναφερθούμε σε αυτήν την παρουσία απλώς με την χρήση του «**application.**». Όταν καλούνται αυτές οι μέθοδοι, πρέπει να περαστεί ως όρισμα είτε μια κλάση είτε μια «παρουσία». Όταν καλείται η μέθοδος

«**addController**», όπου δηλώνουμε τα μέρη της εφαρμογής που αντιστοιχίζουν τις «αλλαγές-hash» στους «διαχειριστές συμβάντων» (**event handlers**), πρέπει να περαστεί ως όρισμα μια «παρουσία» ενός **Ελεγκτή** (**Controller**). Δεν υπάρχει νόημα να μιλάμε για την έννοια «κλάση» για έναν **Ελεγκτή**, επειδή οι **Ελεγκτές** πρέπει να υπάρχουν καθ' όλη την διάρκεια εκτέλεσης της εφαρμογής καθώς ανά πάσα στιγμή μπορεί να γίνει μια «αλλαγή-hash» και να πρέπει να ειδοποιηθεί ο κατάλληλος «διαχειριστής συμβάντος». Οπότε όταν αναφερόμαστε σε **Ελεγκτές** της εφαρμογής, εννοούμε πάντα «παρουσίες» αυτών των **Ελεγκτών**, ή μπορούν να θεωρηθούν και ως «**singleton**» της εφαρμογής, δηλαδή «παρουσίες» οι οποίες υπάρχουν για όλη την διάρκεια της εφαρμογής.

Όταν πρόκειται για προσθήκη **Μοντέλων** ή **Προβολών** στην εφαρμογή, αυτή μπορεί να αναφέρεται και σε «παρουσία» αλλά και σε κλάση. Εάν προστεθεί μια «παρουσία» μιας κλάσης **Μοντέλου** ή ως **Προβολής**, το «**application.js**» θα το θεωρήσει ως ένα «**singleton**» και θα το αποθηκεύσει στην μνήμη (**cache**) ως ένα «**singleton**». Εάν όμως προστεθεί ως ορισμός κλάσης, το «**application.js**» θα «καταλάβει» ότι πρόκειται για μία «παροδική κλάση» (**transient class**) και όταν αργότερα ζητηθεί, η εφαρμογή θα ξέρει ότι πρέπει να την «συγκεκριμενοποιήσει» πρώτα και μετά να την επιστρέψει, διότι πρέπει να δημιουργηθεί μια «παρουσία» της κλάσεις προτού χρησιμοποιηθεί.

Για την συλλογή πληροφορίας, δηλαδή για την συλλογή των **Μοντέλων** και των **Προβολών** της εφαρμογής, χρησιμοποιούνται δυο μέθοδοι:

- application.**getModel**(className [, [arguments]]);
- application.**getView**(className [, [arguments]]);

Δεν υπάρχει έννοια μιας μεθόδου «**getController**», καθώς ο **Ελεγκτής** δουλεύει άμεσα με την εφαρμογή οπότε δεν υπάρχει νόημα στο να συλλέξουμε κάτι από αυτόν. Όμως τα **Μοντέλα** και τις **Προβολές** χρειάζεται να έχουμε την δυνατότητα να τα/τις συλλέξουμε. Όταν καλείται μια από τις δύο παραπάνω μεθόδους, πρέπει να ορίζεται ένα όνομα κλάσης (**className**), δηλαδή μια συμβολοσειρά κειμένου με το όνομα της κλάσης, και ανάλογα με τον τύπο που αυτές οι **Προβολές**/**Μοντέλα** έχουν δηλωθεί, θα μας επιστραφεί είτε μια «παρουσία» «**singleton**» που έχει αποθηκευτεί στην μνήμη (**cached**) ή μια νέα «παρουσία» μιας «παροδικής κλάσης». Εάν σαν όρισμα στις μεθόδους προσθήκης έχει δηλωθεί μια κλάση, και θελήσουμε να την συλλέξουμε, το «**application.js**» θα «καταλάβει» ότι πρόκειται για μια «παροδική παρουσία» (**transient instance**) και θα μας επιτρέψει να περάσουμε ένα δεύτερο όρισμα, το οποίο θα είναι μια σειρά (**array**) από «ορίσματα κατασκευαστές» (**constructor arguments**) τα οποία θα χρειαστούμε για αυτήν την «παροδική συγκεκριμενοποίηση». Για καλύτερη κατανόηση, το θέμα θα αναφερθεί ξανά όταν περάσουμε στο κώδικα της εφαρμογής.

5.1.3 Προσθήκη Ελεγκτών

```
window.application.addController((function( $, application ){
    function Controller(){
        this.route( "/", this.index );
        this.route( "/contacts/edit/:id", this.editContact );
    };
    Controller.prototype = new application.Controller();
    Controller.prototype.init = function(){
        // ...
    };
    Controller.prototype.editContact = function( event, id){
        // ...
    };
    Controller.prototype.index = function( event ){
        // ...
    };
    // ------------------------------------------------ //
    // ------------------------------------------------ //
    return( new Controller() );
})( jQuery, window.application ));
```

Όπως αναφέρθηκε και παραπάνω, όταν προσθέτουμε ένα **Ελεγκτή**, χρησιμοποιούμε την μέθοδο «**application.addController**» όπως φαίνεται και στον παραπάνω κώδικα. Όλο το παραπάνω μπλοκ κώδικα, εκτελείται ως ένα αυτό-εκτελούμενο μπλοκ κώδικα, όπως αναφέρθηκε και παραπάνω. Όπως φαίνεται και στον κώδικα, έχουμε μια ανώνυμη συνάρτηση η οποία αυτό-εκτελείται στο τέλος του μπλοκ όπου δηλώνονται δύο ορίσματα, το ένα είναι η βιβλιοθήκη **jQuery** και το άλλο είναι το «**window.application**». Στο επάνω μέρος όπου ορίζεται η ανώνυμη συνάρτηση δηλώνονται ως παράμετροι το «**$**» και το «**application**». Με αυτόν τον τρόπο ουσιαστικά καταφέρνουμε δυο πράγματα, το ένα είναι ότι όλη αυτή η ανώνυμη συνάρτηση πακετάρεται σε μια δική της «φούσκα» μνήμης, και το δεύτερο είναι ότι αντιστοιχίζουμε το **jQuery** με το «**$**». Το πρώτο εξηγήθηκε παραπάνω γιατί είναι σημαντικό. Το δεύτερο σημαίνει ότι με τον τρόπο που θα εκτελεστεί αυτός ο κώδικας, ανεξάρτητα με το εάν η γενική εφαρμογή έχει ενσωματωμένη υποστήριξη για το **jQuery** ή όχι, ο κώδικας μέσα σε αυτό το μπλοκ μπορεί να χρησιμοποιεί το «**$**» για να αναφέρεται στο **jQuery**.

Μέσα στο μπλοκ κώδικα της συνάρτησης, ορίζουμε την κλάση του **Ελεγκτή**. Ο **Ελεγκτής** εδώ ονομάζεται απλά «**Controller**». Προς το τέλος του κώδικα επιστρέφεται αυτό που θα οριστεί ως όρισμα για την μέθοδο «**addController**» που έχουμε στην αρχή. Το όρισμα αυτό είναι μια νέα «παρουσία» του **Ελεγκτή** «**Controller**» που κατασκευάσαμε με αυτήν την συνάρτηση. Δηλαδή με λίγα λόγια, δημιουργούμε μια νέα κλάση **Ελεγκτή** μέσα στην αυτό-εκτελούμενη συνάρτηση και την προσδίδουμε απευθείας ως μια νέα «παρουσία» **Ελεγκτή** με την χρήση της μεθόδου «**addController**». Το πιο σημαντικό εδώ είναι το γεγονός ότι, επειδή όλο αυτό το μπλοκ κώδικα εκτελείται εντός ενός δικού του πεδίου μνήμης, δεν χρειάζεται να

ανησυχούμε για το εάν το όνομα «**Controller**» που δώσαμε στον **Ελεγκτή** μας συγκρούεται με κάποιο άλλο ίδιο όνομα στην υπόλοιπη εφαρμογή, επειδή το μπλοκ αυτό δεν έχει σχέση με το πεδίο μνήμης του «**global namespace**». Το «**Controller**» δηλαδή γίνεται μια ιδιωτική μεταβλητή, μέσα στην «φούσκα» μνήμης της συνάρτησής μας. Αυτό το γεγονός είναι σημαντικό για μια γλώσσα όπως η **JavaScript**, η οποία εκτελείται χωρίς να «μεταγλωττίζεται» (**compiled**) προηγουμένως. Η **JavaScript** εκτελείται κάτω από το ίδιο «**παράθυρο**», και επομένων, ακόμη και αν είχαμε τα αρχεία με τον κώδικα της σε διαφορετικά σημεία, εάν είχαμε κοινά ονόματα μεταβλητών μεταξύ των αρχείων, θα είχαμε σίγουρα συγκρούσεις των ονομάτων (**name conflicts**).

Οι κατασκευή των **Ελεγκτών** είναι κάπως ειδική περίπτωση, επειδή πρέπει να «επεκτείνουν» μια έκδοση του «**application.Controller**», μιας κλάσης του «**application.js**» για τους **Ελεγκτές**, χρησιμοποιώντας την μέθοδο «**prototype**». Η επέκταση «**prototype**» εδώ είναι κάτι σαν την «**class**» επέκταση, οπότε μπορούμε να θεωρήσουμε την κλάση «**application.Controller**» κατά κάποιο τρόπο ως την υπερκλάση της κλάσης του **Ελεγκτή** που ορίζουμε σε αυτό το μπλοκ. Η επέκταση αυτή απαιτείται γιατί μας δίνει πρόσβαση στην μέθοδο «**this.route**» που χρησιμοποιούμε. Η μέθοδος «**this.route**» είναι αυτή που μας βοηθάει να αντιστοιχίζουμε «αλλαγές-hash» με τους κατάλληλους διαχειριστές συμβάντων, όπως στην περίπτωση αυτή, όπου αντιστοιχίζουμε την τιμή «**/contacts/edit/:id**» της «αλλαγής-hash» με τον διαχειριστή συμβάντος «**this.editContact**».

Όλα τα «εξαρτήματα» (**components**) εντός της εφαρμογής, είτε αυτά είναι **Ελεγκτές**, είτε είναι **Προβολές** ή **Μοντέλα**, έχουν μια μέθοδο «**.init**», και η διαφορά μεταξύ αυτής της μεθόδου και της μεθόδου κατασκευαστή (**constructor method**) είναι ότι η μέθοδος «**.init**» καλείται όταν η εφαρμογή ξεκινάει. Σύμφωνα με την αρχιτεκτονική που έχει κατασκευαστεί η εφαρμογή, η κλήση του κατασκευαστή σε αυτό το μπλοκ κώδικα γίνεται στο τέλος, με την «**return(new Controller());**». Σε εκείνο το σημείο όμως, δεν γνωρίζουμε και πολλά για την κατάσταση που βρίσκεται η εφαρμογή. Είναι το μοντέλο **DOM** έτοιμο; Έχουν προστεθεί τα άλλα «εξαρτήματα» (**Προβολές, Μοντέλα**) στην εφαρμογή; Δεν γνωρίζουμε αν όλα αυτά έχουν γίνει, επειδή η εφαρμογή βρίσκεται στην φάση εκκίνησης. Η μέθοδος κατασκευαστής όμως θα εκτελεστεί αμέσως οπότε θα υπάρξει πρόβλημα. Αυτό το πρόβλημα λύνει η μέθοδος «**init**» η οποία καλείται να εκτελεστεί μόλις η εφαρμογή έχει τελειώσει την αρχική εκκίνηση. Σε εκείνο το σημείο γνωρίζουμε ότι το μοντέλο **DOM** έχει δημιουργηθεί και είναι έτοιμο προς αλληλεπίδραση, όπως επίσης γνωρίζουμε ότι όλα τα «εξαρτήματα» (**Ελεγκτές, Μοντέλα, Προβολές**) έχουν προστεθεί στο σύστημα. Οπότε η μέθοδος «**init**» μπορεί να θεωρηθεί ως μια δεύτερη φάση φόρτωσης της εφαρμογής. Αυτή η μέθοδος δεν απαιτείται και θα εκτελεστεί μόνο εάν υπάρχει εντός της κλάσης.

5.1.3.1 Πως λειτουργεί η δρομολόγηση

Η μέθοδος «**this.route**» δέχεται δύο ορίσματα, το ένα είναι η τιμής της «αλλαγής-hash» και το άλλο είναι ο εσωτερικός διαχειριστής συμβάντος ο οποίος αντιστοιχίζεται με αυτήν την «αλλαγή-hash». Οπότε στο παραπάνω παράδειγμα κώδικα, κατά την πρώτη κλήση της μεθόδου «**this.route**», η τιμή «/» της «αλλαγής-hash» αντιστοιχίζεται με τον διαχειριστή συμβάντος «**this.index**» ο οποίος ορίζεται παρακάτω στην κλάση μας. Ο «**this.index**» λοιπόν είναι απλά μια μέθοδος της κλάσης μας, η οποία δέχεται σαν όρισμα το περιεχόμενο του συμβάντος (**event**). Στην δεύτερη κλήση της μεθόδου «**this.route**», η τιμή της «αλλαγής-hash» είναι «**/contacts/edit/:id**». Το «**:id**» σε αυτήν την περίπτωση, το οποίο είναι ο αριθμός αναγνωριστικό (**id number**) της δεδομένης επαφής, έχει μπροστά του τον χαρακτήρα «**:**», και με αυτόν τον τρόπο γίνεται ένα είδος μεταβλητής της διεύθυνσης **URL**, η οποία παίρνει την τιμή του αναγνωριστικού αριθμού της εκάστοτε επαφής. Έτσι το «**:id**» μπορεί να είναι «**3**», «**123**» ή ότι αριθμός έχει αντιστοιχιστεί με την επαφή που αλληλεπιδρά η εφαρμογή κάθε δεδομένη στιγμή (εάν και εφόσον, αλληλεπιδρά με κάποια). Επομένως το «:id» γίνεται από μεταβλητή της διεύθυνσης **URL** σε μεταβλητή της εφαρμογής. Όποια λοιπόν «αλλαγή-hash» ταιριάξει με το πρότυπο «**/contacts/edit/:id**» (π.χ. η «**/contacts/edit/234**») θα ωθήσει την μέθοδο «**this.route**» να καλέσει τον διαχειριστή συμβάντων «**this.editContact**», ο οποίος με την σειρά του είναι επίσης μια μέθοδος της κλάσης μας και ορίζεται παρακάτω στον κώδικα του μπλοκ παρόμοια με την «**this.index**». Σε αυτήν την περίπτωση όμως, η μέθοδος «**this.editContact**» παίρνει και ένα δεύτερο όρισμα, που δεν είναι άλλο από την μεταβλητή «**id**». Φυσικά η μεταβλητή «**id**» είναι ήδη μέρος του περιεχομένου του συμβάντος ως μέρος της διεύθυνσης **URL**, όμως για ευκολία χρήσης ορίζεται σαν ξεχωριστό όρισμα.

5.1.4 Προσθήκη Μοντέλων (και Προβολών)

```
window.application.addModel((function( $, application ){
    function Controller(){
        // ...
    };
    Controller.prototype = new application.Controller();
    MyClass.prototype.init = function(){
        // ...
    };
    // ------------------------------------------------ //
    // ------------------------------------------------ //
    return( new MyClass() ); // or return( MyClass );
})( jQuery, window.application ));
```

Η προσθήκη ενός **Μοντέλου** ή μιας **Προβολής** είναι πολύ παρόμοια με αυτή ενός **Ελεγκτή**. Αντί να καλείται η μέθοδος «**.addController**» καλούνται οι μέθοδοι «.**addModel**» ή «**.addView**», ανάλογα την περίπτωση. Γενικά, το μπλοκ αυτού του κώδικα εκτελείται με τον

ίδιο τρόπο (δηλαδή της αυτό-εκτελούμενης «φούσκας») και έχει τα ίδια χαρακτηριστικά με την περίπτωση της προσθήκης του **Ελεγκτή** που αναλύσαμε προηγουμένως. Η κλάση μας ορίζεται και εδώ με την ίδια μεθοδολογία. Η διαφορά είναι ότι στην περίπτωση των **Μοντέλων** και των **Προβολών**, υπάρχει η δυνατότητα εκτός από την επιστροφή μια νέας «παρουσίας» της κλάσης («**return(new MyClass());**») δηλαδή ενός «**singleton**», και η δυνατότητα επιστροφής ενός ορισμού κλάσης («**class definition)(«return(MyClass);**») για «παροδικά αντικείμενα» (**transient objects**).

5.1.5 Η κεντρική σελίδα «index.html»

Έχοντας αναλύσει τις τεχνικές συγγραφής του κώδικα και τις βασικές μεθόδους με την χρήση τους, μπορούμε να περάσουμε σε ενδεικτικά αποσπάσματα κώδικα της εφαρμογής. Η κεντρική σελίδα της εφαρμογής είναι η «**index.html**» και καθώς πρόκειται για μία «**μονοσέλιδη**» εφαρμογή, η σελίδα «**index.html**» ουσιαστικά **είναι** η εφαρμογή. Ολόκληρος ο κώδικας της σελίδας φαίνεται στις επόμενες σελίδες. Αφήνοντας τις δηλώσεις των «**scripts**», στις οποίες θα αναφερθούμε αργότερα, και βλέποντας το κεντρικό σώμα του κώδικα, παρατηρούμε ότι πρόκειται για ένα σχετικά απλό κομμάτι κώδικα το οποίο ουσιαστικά απαρτίζεται από ετικέτες «** **» οι οποίες εσωκλείουν λίστες, φόρμες, πίνακες και άλλα γνωστά στοιχεία της **HTML** γλώσσας . Υπάρχει στην αρχή του κώδικα το κεφάλι της σελίδας, με το λογότυπο και την κόκκινη ειδοποίηση «**Φόρτωση…**». Αυτή η ειδοποίηση είναι μια **AJAX** ειδοποίηση, η οποία εμφανίζεται κάθε φορά που έχουμε μια **AJAX** επικοινωνία σε εξέλιξη στο παρασκήνιο. Στην συνέχεια έχουμε το κυρίως σώμα της εφαρμογής, το οποίο εσωτερικά είναι μοιρασμένο σε δύο «**διατάξεις**» (**layouts**).

Η μία «διάταξη» αποτελεί την λίστα επαφών («**contact-list-layout**») η οποία αποτελείται από την φόρμα αναζήτησης στην κορυφή, την σύνδεση «νέαεπαφή» και φυσικά την λίστα με τις επαφές («**<ul id="contact-list">**») η οποία όμως είναι αρχικά άδεια και πρόκειται να «γεμίσει» δυναμικά με την εκκίνηση της εφαρμογής. Κάτω από την λίστα έχουμε ένα «πρότυπο» για τα αντικείμενα της λίστας («**contact-list-item-template**») το οποίο είναι μια ετικέτα «**script**» που μέσα της περιέχει στοιχεία λίστας τα οποία πρόκειται και αυτά να δημιουργηθούν από το «πλαίσιο» της εφαρμογής, βάση των δεδομένων που θα ανακτώνται από τον Διακομιστή. Βλέπουμε ότι ο τύπος της ετικέτας «**script**» είναι «**type="application/template**"» και ότι τα στοιχεία τις λίστας αποτελούνται από το «**$**» και εντός των αγκύλων το όνομα της ιδιότητας (δηλαδή **name**, **phone** και **email**).

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8"/>
    <title>Εξυπνες Επαφές</title>
    <link rel="stylesheet" type="text/css"
href="./linked/css/reset.css"></link>
    <link rel="stylesheet" type="text/css"
href="./linked/css/standard.css"></link>
    <script type="text/javascript" src="./linked/js/app/jquery-
1.4.2.min.js"></script>
    <script type="text/javascript"
src="./linked/js/app/application.js"></script>
    <!--- Controllers. --->
    <script type="text/javascript"
src="./linked/js/controller/contacts.js"></script>
    <!--- Model. --->
    <script type="text/javascript"
src="./linked/js/model/contact_service.js"></script>
    <script type="text/javascript"
src="./linked/js/model/contact.js"></script>
    <!--- View. --->
    <script type="text/javascript"
src="./linked/js/view/ajax_notification.js"></script>
    <script type="text/javascript"
src="./linked/js/view/contact_list.js"></script>
    <script type="text/javascript"
src="./linked/js/view/contact_form.js"></script>
    <script type="text/javascript"
src="./linked/js/view/scrolltopcontrol.js"></script>
</head>
<body>
    <!-- BEGIN: Site Wrapper. -->
    <div id="site-wrapper">
        <!-- BEGIN: Site Header. -->
        <div id="site-header" href="#top">
            <a id="site-logo" href="#/">Εξυπνες Επαφές</a>
            <div id="ajax-load-notification">Φόρτωση...</div>
        </div>
        <!-- END: Site Header. -->
        <!-- BEGIN: Site Body. -->
        <div id="site-body">
            <!-- BEGIN: Content Layouts. -->
            <ul id="content-layouts">
                <!-- BEGIN: Content Layout. -->
                <li id="contact-list-layout" class="content-
layout" style="display: none ;">
                    <div id="contact-list-header" class="content-
header">
                        <form>
                            <label>αναζήτηση:</label>
                            <input type="text" />
                        </form>
                        <a href="#/contacts/add/">
                            <span
class="contrast">νέα</span>επαφή &raquo;
                        </a>
                    </div>
```

```
                    <div id="contact-list-body" class="content-
body">
                        <ul id="contact-list">
                            <!-- To be filled dynamically. -->
                        </ul>
                    </div>
                    <!-- BEGIN: Templates. -->
                    <script id="contact-list-item-template"
type="application/template">
                        <li class="contact clear-fix">
                            <div class="summary">
                                <a class="name">${name}</a>
                            </div>
                            <div class="actions">
                                <a href="javascript:void( 0 )"
class="more">περισσ.</a>  | 
                                <a href="#/contacts/edit/${id}"
class="edit">επεξ.</a>  | 
                                <a href="#/contacts/delete/${id}"
class="delete">διαγραφή</a>
                            </div>
                            <dl class="details clear-fix">
                                <dt>
                                    όνομα:
                                </dt>
                                <dd>
                                    ${name}
                                </dd>
                                <dt>
                                    τηλ.:
                                </dt>
                                <dd>
                                    ${phone}
                                </dd>
                                <dt>
                                    email:
                                </dt>
                                <dd>
                                    ${email}
                                </dd>
                            </dl>
                        </li>
                    </script>
                    <!-- END: Template. -->
                </li>
                <!-- END: Content Layout. -->
                <!-- BEGIN: Content Layout. -->
                <li id="contact-edit-layout" class="content-
layout" style="display: none ;">
                    <div id="contact-edit-header" class="content-
header">
                        <a href="#/">&laquo; πίσω</a>
                    </div>
                    <div id="contact-edit-body" class="content-
body">
                        <form id="contact-form">
                            <input type="hidden" name="id"
value="0" />
```

```
                            <ul class="errors">
                                <!-- To be populated later. -->
                            </ul>
                            <div class="entry">
                                <label>
                                    Όνομα:
                                </label>
                                <input type="text" name="name" />
                            </div>
                            <div class="entry">
                                <label>
                                    Τηλέφωνο:
                                </label>
                                <input type="text" name="phone"
/>
                            </div>
                            <div class="entry">
                                <label>
                                    Email:
                                </label>
                                <input type="text" name="email"
/>
                            </div>
                            <div class="actions">
                                <button type="submit">Αποθήκευση
&amp;amp; Κλείσιμο</button>
                                ( <a href="#/contacts"
class="cancel">Ακύρωση</a> )
                            </div>
                        </form>
                    </div>
                </li>
                <!-- END: Content Layout. -->
            </ul>
            <!-- END: Content Layouts. -->
        </div>
        <!-- END: Site Body. -->
        <!-- BEGIN: Site Footer. -->
        <div id="site-footer">
            <div id="site-copyright">
                <a href="" target="_blank">Τυμπακιανάκης
Απόστολος</a>  2010
            </div>
        </div>
        <!-- END: Site Footer. -->
    </div>
    <!-- END: Site Wrapper. -->
</body>
</html>
```

Η δεύτερη διάταξη είναι η διάταξη της φόρμας επεξεργασίας-δημιουργίας επαφών («**contact-edit-layout**»). Όπως έχει αναφερθεί και σε προηγούμενη ενότητα, η φόρμα δημιουργίας και η φόρμα επεξεργασίας επαφής είναι ουσιαστικά η ίδια φόρμα, η οποία είναι άδεια όταν προσθέτουμε και είναι «γεμάτη» με τα στοιχεία της επαφής όταν επεξεργαζόμαστε. Είναι μια απλή δομή φόρμας με λίστες και άλλα γνωστά στοιχεία της **HTML**.

Ουσιαστικά αυτή είναι η κεντρική σελίδα, δεν πρόκειται για μια περίπλοκη σελίδα, είναι πολύ απλή η δομή και τα στοιχεία της, καθώς την κύρια δουλεία στην εφαρμογή την κάνει ο κώδικας της **JavaScript** και κατ' επέκταση η βιβλιοθήκη **jQuery**. Βλέποντας λοιπόν τις δηλώσεις στην αρχή του κώδικα, βλέπουμε ότι δηλώνονται χωριστά τα «**scripts**» για τους **Ελεγκτές** (**controllers**) για τα **Μοντέλα** (**Models**) και τις **Προβολές** (**Views**). Αυτό έγινε για ευκολία κατανόησης και για ευκολία ανάπτυξης και αποσφαλμάτωσης των επιμέρους μελών της εφαρμογής. Σε πραγματικά περιβάλλοντα ανάπτυξης, ένας προγραμματιστής πρέπει να συγχωνεύσει αυτά τα «**scripts**» σε όσα λιγότερα γίνεται, έτσι ώστε να μειωθούν οι αιτήσεις αρχείων και επομένως ο φόρτος εργασίας προς τον Διακομιστή. Οι δηλώσεις ξεκινάνε με τα αρχεία φύλλων στυλ «**CSS**» τα οποία αναλαμβάνουν σχεδόν εξ' ολοκλήρου την αισθητική εμφάνιση της εφαρμογής γενικότερα. Στην συνέχεια δηλώνονται η βιβλιοθήκη **jQuery** καθώς και το κεντρικό αρχείο της εφαρμογής «**application.js**» τα οποία αποτελούν τον πυρήνα της εφαρμογής. Έπειτα δηλώνονται οι **Ελεγκτές**, οι **Προβολές** και τα **Μοντέλα** της εφαρμογής και με βάση όσα έχουμε δει μέχρι τώρα, κάθε ένα από αυτά τα αρχεία αποτελεί και μια δική του κλάση, και όλα έχουν συνταχτεί με παρόμοιο τρόπο, ο οποίος περιγράφεται στις προηγούμενες παραγράφους της ενότητας. Στις επόμενες παραγράφους θα περιγραφτούν ενδεικτικά κάποιες από αυτές τις κλάσεις.

5.1.6 Το Μοντέλο «contact.js»

```
window.application.addModel((function( $, application ){
    function Contact( id, name, phone, email ){
        this.id = (id || 0);
        this.name = (name || "");
        this.phone = (phone || "");
        this.email = (email || "");
    };
    Contact.prototype.validate = function(){
        return( [] );
    };
    // Return a new model class.
    return( Contact );
})( jQuery, window.application ));
```

Η περιγραφή των κλάσεων ξεκινάει από το **Μοντέλο**, καθώς έχει το μικρότερο βαθμό αλληλεπίδρασης και έτσι είναι πιθανότατα το πιο απλό κομμάτι της εφαρμογής. Αρχίζουμε με το αρχείο «**contact.js**» το οποίο είναι και το πιο απλό από τα **Μοντέλα**. Ο κώδικας του αρχείου φαίνεται παρακάτω. Εδώ έχουμε την κλάση «**Contact**» η οποία κατασκευάζεται και στην συνέχεια επιστρέφεται ως όρισμα στην μέθοδο «**application.addModel**» με τον τρόπο που περιγράψαμε σε προηγούμενη παράγραφο. Η κλάση «**Contact**» είναι μια αντιπροσώπευση μιας επαφής του συστήματος, η οποία αποτελείται από το αναγνωριστικό (**id**), το όνομα (**name**), τον αριθμό τηλέφωνου (**phone**) και την διεύθυνση ηλεκτρονικού

ταχυδρομείου (**email**) της επαφής. Για λόγους απλότητας δεν περιέχονται μέθοδοι για την πρόσβαση και διαχείριση αυτών των ιδιοτήτων (**properties**), και έτσι για να γίνει αυτό θα πρέπει να κληθεί μια δημόσια ιδιότητα για την επαφή. Η μέθοδος «**validate**» υπάρχει για την επικύρωση της επαφής, δεν χρησιμοποιείται όμως καθώς η επικύρωση γίνεται πάντα από την πλευρά του Διακομιστή, καθώς με την εξέλιξη της ανάπτυξης της εφαρμογής ο κύριος φόρτος αυτών των εργασιών μεταφέρθηκε στον Διακομιστή. Έτσι για παράδειγμα όταν μια επαφή πρέπει να επικυρωθεί για να αποθηκευτεί στην συνέχεια, αυτή τη ενέργεια «αποθήκευσης» στέλνεται στον Διακομιστή, και αυτός αναλαμβάνει την επικύρωση της επαφής και στην συνέχεια την αποθήκευσή της.

Όπως φαίνεται από την επιστροφή της συνάρτησης στο τέλος του κώδικα, δεν επιστρέφεται μια «παρουσία» της κλάσης αλλά μία «παροδική κλάση» της «**Contact**». Αυτό γίνεται επειδή χρειαζόμαστε μια «παρουσία» της κλάσης για κάθε επαφή που υπάρχει, είτε αυτές είναι είκοσι στην λίστα, είτε είναι απλά μία στην λίστα. Οπότε με το να «επιστρέφεται» μόνο το όνομα της κλάσης («**return(Contact);**») δηλώνουμε στην εφαρμογή πως κάθε ακόλουθη αίτηση της κλάσης αυτής θα απαιτεί «συγκεκριμενοποίηση» μια νέας «παρουσίας» αυτής της «παροδικής κλάσης».

5.1.7 Το Μοντέλο «contact_service.js»

Το **Μοντέλο** «**contact_service.js**» είναι λίγο πιο περίπλοκο από το προηγούμενο **Μοντέλο**, όμως κινείται στις ίδιες γενικές γραμμές και ακολουθεί τις ίδιες τακτικές σύνταξης. Έχουμε την μέθοδο «**.addModel**» στην κορυφή η οποία θα δεχτεί σαν όρισμα την κλάση που θα επιστρέψει η αυτό-εκτελούμενη συνάρτηση. Στην ανάλυση της αρχιτεκτονικής της πλευράς του Πελάτη, αναφέρθηκε ότι το μόνο μέρος της πλευράς αυτής που επικοινωνεί με τον Διακομιστή είναι το **Μοντέλο**. Εδώ μπορούμε να πούμε και συγκεκριμένα, πως η κλάση «**contact_service.js**» είναι το μέρος του **Μοντέλου** που αναλαμβάνει την όποια επικοινωνία απαιτείται με την πλευρά του Διακομιστή.

Κατά κάποια έννοια, το «**contact_service.js**» στην πλευρά του Πελάτη γίνεται ένα αφηρημένο «στρώμα» δεδομένων (**data abstraction layer**) για την συλλογή των δεδομένων των επαφών και την αποθήκευση των δεδομένων αυτών. Ωστόσο, επειδή δεν αποθηκεύουμε δεδομένα στην πλευρά του Πελάτη αλλά τα αποθηκεύουμε στην πλευρά του Διακομιστή, αυτή η κλάση ουσιαστικά ενεργεί ως ένα «περιτύλιγμα» (**wrapper**) για της κλήσεις **AJAX** που θα γίνονται προς τον Διακομιστή για την αποθήκευση και ανάκτηση πληροφορίας.

```
window.application.addModel((function( $, application ){
    function ContactService(){
        // ...
    };
    ContactService.prototype.init = function(){
        // ...
    };
    ContactService.prototype.deleteContact = function( id,
onSuccess, onError ){
        var self = this;
        application.ajax({
            url: "api/Contacts.cfc",
            data: {
                method: "deleteContact",
                id: id
            },
            normalizeJSON: true,
            success: function( response ){
                if (response.success){
                    onSuccess( id );
                } else if (onError){
                    onError( respsonse.errors );
                }
            }
        });
    };
    ContactService.prototype.getContact = function( id,
onSuccess, onError ){
        var self = this;
        application.ajax({
            url: "api/Contacts.cfc",
            data: {
                method: "getContact",
                id: id
            },
            normalizeJSON: true,
            success: function( response ){
                if (response.success){
                    onSuccess( self.populateContactsFromResponse(
response.data ) );
                } else if (onError){
                    onError( respsonse.errors );
                }
            }
        });
    };
    ContactService.prototype.getContacts = function( onSuccess,
onError ){
        var self = this;
        application.ajax({
            url: "api/Contacts.cfc",
            data: {
                method: "getContacts"
            },
            normalizeJSON: true,
            success: function( response ){
                if (response.success){
                    onSuccess( self.populateContactsFromResponse(
response.data ) );
```

```
                } else if (onError){
                    onError( respsonse.errors );
                }
            }
        });
    };
    ContactService.prototype.populateContactsFromResponse =
function( responseData ){
        if ($.isArray( responseData )){
            var contacts = [];
            $.each(
                responseData,
                function( index, contactData ){
                    contacts.push(
                        application.getModel( "Contact", [
contactData.id, contactData.name, contactData.phone,
contactData.email ] )
                    );
                }
            );
            return( contacts );
        } else {
            return(
                application.getModel( "Contact", [
responseData.id, responseData.name, responseData.phone,
responseData.email ] )
            );
        }
    };
    ContactService.prototype.saveContact = function( id, name,
phone, email, onSuccess, onError ){
        var self = this;
        application.ajax({
            url: "api/Contacts.cfc",
            data: {
                method: "saveContact",
                id: id,
                name: name,
                phone: phone,
                email: email
            },
            normalizeJSON: true,
            success: function( response ){
                if (response.success){
                    onSuccess( response.data );
                } else {
                    onError( response.errors );
                }
            }
        });
    };
    return( new ContactService() );
})( jQuery, window.application ));
```

Όπως και όλες οι άλλες κλάσεις έχει την μέθοδο «**init**» για να αρχικοποιείται όταν εκκινεί η εφαρμογή, αν και σε αυτήν την περίπτωση δεν χρειάζεται να γίνει κάτι τέτοιο. Στον υπόλοιπο κώδικα υπάρχουν οι τέσσερις μέθοδοι, «**deleteContact**», «**getContact**», «**getContacts**» και «**saveContact**», οι οποίες περιλαμβάνουν αντίστοιχα τέσσερις συναρτήσεις που εκτελούν τις

κλήσεις προς τον Διακομιστή («**application.ajax**»). Όπως δηλώνουν και τα ονόματα, οι τέσσερις αυτές μέθοδοι αναλαμβάνουν τα εξής:

- **deleteContact** --> διαγραφή μιας επαφής
- **getContact** --> ανάκτηση μιας επαφής
- **getContacts** --> ανάκτηση μιας λίστας από επαφές
- **saveContact** --> αποθήκευση μιας νέας ή ενημέρωση μιας υπάρχουσας επαφής

Αυτές οι μέθοδοι είναι ο πυρήνας της επικοινωνίας του Πελάτη με τον Διακομιστή, και όλες λειτουργούν με παρόμοιο τρόπο. Για παράδειγμα η μέθοδος «**deleteContact**», παίρνει αρχικά ως όρισμα το αναγνωριστικό (**id**) της επαφής που της ζητείται να διαγράψει. Επειδή το αφηρημένο στρώμα δεδομένων απαιτεί την επικοινωνία με τον Διακομιστή μέσω **AJAX**, δηλαδή με ασύγχρονο τρόπο, δεν μπορούμε απλά να επιστρέψουμε πληροφορία από αυτήν την μέθοδο γιατί δεν έχουμε αμέσως αυτήν την πληροφορία. Εξηγώντας, λόγω της ασύγχρονης φύσης της επικοινωνίας, η κάθε μέθοδος που χρησιμοποιεί κλήσεις **AJAX** μετά την εκτέλεσή της δεν σταματάει την συνολική εκτέλεση της εφαρμογής αλλά περιμένει στο παρασκήνιο την απάντηση των δεδομένων που αιτείται. Υπάρχει λοιπόν η περίπτωση αυτή η μέθοδος να τελειώσει την εκτέλεσή της, πριν καν είναι διαθέσιμα τα ζητούμενα δεδομένα. Γι αυτούς τους παραπάνω λόγους, όταν εκτελείται μια τέτοιου είδους μέθοδος, δεν δέχεται σαν ορίσματα μόνο τα δεδομένα που χρειάζεται για την εργασία που καλείται να εκτελέσει (δηλαδή στην περίπτωσή αυτή το «**id**») αλλά δέχεται άλλα δύο επιπλέον ορίσματα, «**onSuccess**» και «**onError**», δηλαδή τις επανακλήσεις (**callbacks**) για την περίπτωση επιτυχής εκτέλεσης της ενέργειας (π.χ. την επιτυχή διαγραφή της επαφής) ή της μη επιτυχής εκτέλεσης εξ' αιτίας κάποιου λάθους.

Προχωρώντας στον κώδικα της «**deleteContact**» βλέπουμε την μέθοδο «**application.ajax**» η οποία ουσιαστικά δίνει εντολή στην μέθοδο «**ajax**» της βιβλιοθήκης του **jQuery** να εκτελεστεί. Η μέθοδος αυτή κάνει μια κλήση για την απομακρυσμένη διεπαφή **API** («**url: "api/Contacts.cfc"**») που θα εκτελέσει την επιθυμητή ενέργεια, δηλαδή σε αυτήν την περίπτωση την μέθοδο «**deleteContact**» της διεπαφής αυτής («**method: "deleteContact"**»). Επίσης στέλνεται το αναγνωριστικό της επαφής («**id: id**») που θα διαγραφεί, και δηλώνουμε στην αίτηση να ομαλοποιήσει το **JSON** («**normalizeJSON: true**»), το οποίο θα αναλυθεί λίγο παρακάτω. Στην συνέχεια και εάν η αίτηση επιστρέψει με επιτυχία, ελέγχεται η απάντηση για το ένα υπήρξε επιτυχής ή όχι. Εάν η απάντηση ήταν επιτυχείς, θα κληθεί η επανάκληση «**onSuccess**» η οποία θα επιστρέψει το αναγνωριστικό της επαφής που διαγράφτηκε («**onSuccess(id);**»). Εάν η απάντηση υπήρξε ανεπιτυχής, θα κληθεί η επανάκληση «**onError**» και θα επιστρέψει την συλλογή με τα λάθη που οδήγησαν σε αυτήν την αποτυχία («**onError(respsonse.errors);**»).

Κοιτάζοντας και τις υπόλοιπες μεθόδους, εύκολα παρατηρεί κανείς ότι εκτελούνται με τον ίδιο τρόπο. Όλες εκτελούν κλήσεις **AJAX** και όλες διαθέτουν τους ίδιους διαχειριστές «**onSuccess**» και «**onError**». Όσον αφορά την εντολή «**normalizeJSON: true**», επειδή η εφαρμογή αυτή αναπτύσσεται στα πλαίσια του **ColdFusion**, πρέπει να ληφθούν υπόψη διαφορά χαρακτηριστικά του **ColdFusion** όσον αφορά την σειριοποίηση (**serialization**) των δεδομένων. Επειδή το **ColdFusion** δεν είναι μια εφαρμογή ευαίσθητη στην διάκριση πεζών και κεφαλαίων (**case-sensitive application**) αλλά η γλώσσα **JavaScript** είναι ευαίσθητη στην διάκριση αυτή, αυτή η διαφοροποίηση μπορεί να οδηγήσει σε διάφορα προβλήματα. Έτσι όταν σειριοποιούνται δεδομένα στην πλευρά του Διακομιστή στο πρότυπο **JSON** συχνά παρατηρείται μια κεφαλαιοποίηση των κλειδιών εντός των δομών, το οποίο είναι πρόβλημα.

Για να αντιμετωπιστεί αυτό το πρόβλημα, χωρίς να χρειάζεται η εφαρμογή να «ανησυχεί» για το τι μπορεί να γίνει από την πλευρά του Διακομιστή, προστίθεται η εντολή «**normalizeJSON: true**», η οποία αυτό που κάνει είναι, εσωτερικά της εφαρμογής, όταν η αίτηση **AJAX** επιστραφεί και το **JSON** της απάντησης αναλύεται στα διάφορα αντικείμενα, η εφαρμογή θα σαρώσει αυτά τα αντικείμενα και θα «πεζοποιήσει» όλα τα κλειδιά. Επομένως, και εφόσον δηλώνουμε και τα κλειδιά «**response.success**» και «**respsonse.errors**» της απάντησης με πεζούς χαρακτήρες, η εφαρμογή ενεργεί πάντα υποθέτοντας ότι όλα τα κλειδιά είναι γραμμένα με πεζούς χαρακτήρες. Αυτό το γεγονός επιτρέπει στην εφαρμογή της πλευράς του Πελάτη να μην είναι εξαρτημένη από την τεχνολογία που χρησιμοποιείται στην πλευρά του Διακομιστή, και είναι πολύ σημαντικό καθώς όπως έχει αναφερθεί και στο θεωρητικό μέρος, μια εφαρμογή **AJAX** είναι επικεντρωμένη στην πλευρά του Πελάτη και θα πρέπει να λειτουργεί ακόμη και χωρίς αλλαγές στην τεχνολογία της πλευράς του Διακομιστή.

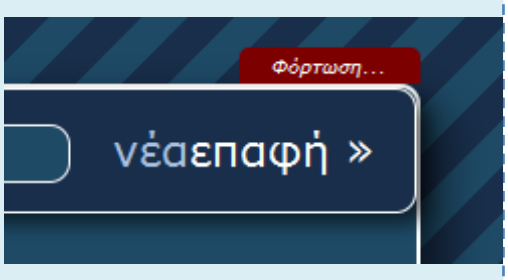
Στην περίπτωση της μεθόδου «**getContact**», τα δεδομένα που επιστρέφει η επανάκληση «**onSuccess**» δεν είναι απλά ένας αριθμός όπως το αναγνωριστικό, αλλά κάτι πιο περίπλοκο («**onSuccess(self.populateContactsFromResponse(response.data));**»). Σε αυτήν την περίπτωση δεν θα επιστραφούν απλά δεδομένα αλλά θα κληθεί η μέθοδος «**self.populateContactsFromResponse**» της κλάσης η οποία ορίζεται παρακάτω στον κώδικα. Αυτή η μέθοδος παίρνει τα δεδομένα που επιστράφηκαν από την απάντηση, τα οποία μπορεί να είναι ένα αντικείμενο **JSON** που αντιστοιχεί σε μία επαφή ή μια σειρά (**array**) από αντικείμενα **JSON** που αντιστοιχούν σε μια λίστα από επαφές, και συμπληρώνει (**populate**) την λίστα με τις επαφές που εμφανίζονται στην εφαρμογή. Αυτό το καταφέρνει καλώντας μια «παρουσία» της κλάσης «**Contact**» της εφαρμογής (μέσω της μεθόδου «**application.getModel**») και δίνοντάς της ως ορίσματα τα στοιχεία της εκάστοτε επαφής (δηλαδή τα **id**, **name**, **phone**, **email**) τα οποία περιέχονται στα δεδομένα της απάντησης που βρίσκονται εντός του εκάστοτε αντικειμένου **JSON.** Γυρίζοντας πίσω στην κλάση «**Contact**»

που περιγράφτηκε πιο πάνω, μπορεί να κατανοηθεί η σύνδεση αυτών των δύο κλάσεων/μεθόδων.

Η κλάση αυτή ολοκληρώνεται με την επιστροφή μιας νέας «παρουσίας» της κλάσης «**ContactService**», η οποία αντιμετωπίζεται από την εφαρμογή ως ένα «**singleton**», εφόσον δεν χρειαζόμαστε περισσότερη από μία «παρουσία» αυτής της κλάσης να υπάρχει ταυτόχρονα στην εφαρμογή μας. Συνοψίζοντας, μπορεί αυτή η κλάση να ήταν πιο περίπλοκη, αλλά εύκολα παρατηρείται ότι ακολουθεί τις ίδιες γενικές γραμμές και τεχνικές όπως περιγράφτηκαν προηγουμένως.

5.1.8 Η προβολή «ajax_notification.js»

```
window.application.addView((function( $, application ){
    function AjaxNotification(){
        this.notification = null;
    };
    AjaxNotification.prototype.init = function(){
        var self = this;
        this.notification = $( "#ajax-load-notification" );
        this.notification.ajaxStart(
            function(){
                self.showView();
            }
        );
        this.notification.ajaxStop(
            function(){
                self.hideView();
            }
        );
        this.hideView();
    };
    AjaxNotification.prototype.hideView = function(){
        this.notification
            .stop()
            .fadeTo( 200, 0 )
        ;
    };
    AjaxNotification.prototype.showView = function(){
        this.notification
            .stop()
            .fadeTo( 200, 1 )
        ;
    };
    return( new AjaxNotification() );
})( jQuery, window.application ));
```



Όταν πρόκειται για τις **Προβολές** της εφαρμογής, η **Προβολή** που θεωρείται πιο απλή είναι η κλάση «**ajax_notification.js**». Πρόκειται για την κλάση που αναλαμβάνει την προβολή της κόκκινης ειδοποίησης που εμφανίζεται στην πάνω δεξιά γωνία του κυρίου μέρους της σελίδας (όπως φαίνεται στην εικόνα) κατά την διάρκεια της εκτέλεσης μιας **AJAX** ασύγχρονης επικοινωνίας στο παρασκήνιο. Η κλάση αυτή ακολουθεί την ίδια δομή με τις

υπόλοιπες, επομένως υπάρχει η μέθοδος «**.addView**» η οποία εσωκλείει μια αυτό-εκτελούμενη ανώνυμη συνάρτηση της οποία την απάντηση παίρνει ως όρισμα. Σε αυτήν την περίπτωση έχουμε επιστροφή μια νέας «παρουσίας» της κλάσης (ενός «**singleton**») η οποία παραμένει και χρησιμοποιείται από την εφαρμογή όταν χρειαστεί.

Στον κατασκευαστή της κλάσης, έχουμε απλά τον ορισμό της ιδιότητας «**this.notification**» ως κενό, εφόσον δεν γνωρίζουμε σε ποια κατάσταση βρίσκεται η εφαρμογή, όπως αναφέρθηκε και παλαιότερα. Μόλις η εφαρμογή είναι έτοιμη για αλληλεπίδραση, η μέθοδος «**init**» εκτελείται και αναλαμβάνει την πραγματική δουλειά αυτής της κλάσης. Με την εκτέλεση της, αρχικά ζητάει από το μοντέλο **DOM** το στοιχείο «**ajax-load-notification**» το οποίο βρίσκεται στην σελίδα «**index.html**» όπως έχει αναφερθεί και προηγουμένως. Επομένως με την μέθοδο «**this.notification = $("#ajax-load-notification");**» θα παρθεί μια συλλογή αναφοράς της βιβλιοθήκης **jQuery** για αυτό το στοιχείο, και θα αποδοθούν στο στοιχείο αυτό οι μέθοδοι παρακολούθησης του **AJAX** («**ajaxStart**» και «**ajaxStop**»), όπως φαίνεται στην συνέχεια του κώδικα. Όταν αυτές οι μέθοδοι κληθούν για εκτέλεση, θα καλέσουν με την σειρά τους τις μεθόδους «**self.showView**» και «**self.hideView**» αντίστοιχα. Όπως είναι φανερό και από τα ονόματα, όταν ξεκινήσει μια κλήση **AJAX**, η μέθοδος «**ajaxStart**» θα εκτελεστεί και θα εμφανίσει την ειδοποίηση της σελίδας με την βοήθεια της μεθόδου «**self.showView**», ενώ αντίστοιχα όταν ολοκληρωθεί η κλήση **AJAX**, η μέθοδος «**ajaxStop**» θα κρύψει την ειδοποίηση της σελίδας με την βοήθεια της μεθόδου «**self.hideView**».

Οι δύο μέθοδοι που αναλαμβάνουν την προβολή και την απόκρυψη της ειδοποίησης, οι «**self.showView**» και «**self.hideView**», ορίζονται παρακάτω στον κώδικα και αποτελούνται από αλλαγές στην ιδιότητα «**this.notification**», που ορίστηκε στον κατασκευαστή, χρησιμοποιώντας το εφέ «**fadeTo**» της βιβλιοθήκης **jQuery** για να εμφανίσουν και να εξαφανίσουν το στοιχείο «**ajax-load-notification**» της σελίδας. Στην τέλος της μεθόδου «**init**» καλείται η μέθοδος «**this.hideView**» για να κρύψει την ειδοποίηση, καθώς δεν θέλουμε η εφαρμογή να εμφανίζει κάποια ειδοποίηση εάν δεν υπάρχει **AJAX** δραστηριότητα στο παρασκήνιο. Αυτή είναι μια απλή κλάση **Προβολής** η οποία λειτουργεί με τον ίδιο τρόπο με τις υπόλοιπες. Στο ίδιο μοτίβο κινούνται όλες οι κλάσεις προβολών στην εφαρμογή.

5.1.9 Ο Ελεγκτής «contacts.js»

```
window.application.addController((function( $, application ){
    function Controller(){
        this.route( "/", this.index );
        this.route( "/contacts/", this.index );
        this.route( "/contacts/add/", this.addContact );
        this.route( "/contacts/edit/:id", this.editContact );
        this.route( "/contacts/delete/:id", this.deleteContact );
        this.currentView = null;
        this.contactListView = null;
        this.contactFormView = null;
    };
    Controller.prototype = new application.Controller();
    Controller.prototype.init = function(){
        this.contactListView = application.getView( "ContactList"
);
        this.contactFormView = application.getView( "ContactForm"
);
    };
    Controller.prototype.addContact = function( event ){
        this.showView( this.contactFormView, event );
    };
    Controller.prototype.editContact = function( event, id ){
        this.showView( this.contactFormView, event );
    };
    Controller.prototype.deleteContact = function( event, id ){
        application.getModel( "ContactService" ).deleteContact(
            id,
            function(){
                application.relocateTo( "contacts" );
            }
        );
    };
    Controller.prototype.index = function( event ){
        this.showView( this.contactListView, event );
    };
    Controller.prototype.showView = function( view, event ){
        if (this.currentView && this.currentView.hideView){
            this.currentView.hideView();
        }
        view.showView( event.parameters );
        this.currentView = view;
    };
    return( new Controller() );
})( jQuery, window.application ));
```

Σε αυτήν την εφαρμογή χρησιμοποιείται μόνο ένας **Ελεγκτής**, η κλάση «**contacts.js**», όμως φυσικά θα μπορούσαν να ήταν περισσότεροι ανάλογα με τον τύπο και τις απαιτήσεις της εκάστοτε εφαρμογής που αναπτύσσεται. Η κλάση αυτή ακολουθεί την ίδια τεχνική ανάπτυξης με τις προηγούμενες με την μέθοδο «**addController**» να προσθέτει στην εφαρμογή της επιστροφή της αυτό-εκτελούμενης συνάρτησης που εμπεριέχεται μέσα της. Μέσα στον κατασκευαστή της κλάσης («**Controller()**») γίνεται η αντιστοίχιση των τιμών των «αλλαγών-hash» με τους διαχειριστές συμβάντων, που έχουν οριστεί παρακάτω στον

κώδικα, οι οποίοι είναι υπεύθυνοι για τις περαιτέρω ενέργειες της εφαρμογής. Παρατηρείται ότι μπορούμε να έχουμε τον ίδιο διαχειριστή για πολλαπλές τιμές «αλλαγών-hash», όπως στην περίπτωση του «**this.index**» ο οποίος αντιστοιχίζεται με τις τιμές «/» και «**/contacts/**». Αυτό μπορεί να συμβεί και αντίστροφα, καθώς η σχέση μεταξύ των τιμών και των διαχειριστών είναι «πολλά προς πολλά».

Μετά την δρομολόγηση των τιμών με τους αντίστοιχούς διαχειριστές με την βοήθεια της μεθόδου «**this.route**», ορίζουμε τις ιδιότητες μας στον κατασκευαστή, οι οποίες για άλλη μια φορά ορίζονται ως κενές, εφόσον δεν γνωρίζουμε εάν η εφαρμογή είναι έτοιμη προς αλληλεπίδραση. Μόλις γίνει το τελευταίο καλείται η μέθοδος «**init**», η οποία αρχικά συλλέγει από την εφαρμογής δύο **Προβολές**, την «**ContactList**» και την «**ContactForm**», οι οποίες αντιπροσωπεύουν τις δύο διατάξεις (**layouts**) της κεντρικής σελίδας της εφαρμογής, όπως περιγράφτηκαν σε προηγούμενη παράγραφο. Η «**ContactList**» αντιστοιχεί στην λίστα επαφών με το πεδίο αναζήτησης και η «**ContactForm**» αντιστοιχεί στην φόρμα επεξεργασίας-δημιουργίας μια επαφής. Ο λόγος που συλλέγονται αυτές οι επαφές εδώ είναι ότι ο συγκεκριμένος **Ελεγκτής** αναλαμβάνει της εναλλαγή αυτών των δύο διατάξεων ανάλογα με τα συμβάντα στην διεύθυνση **URL.**

Σύμφωνα με την μεθοδολογία που περιγράφτηκε σε στην αρχή αυτής της ενότητας, με τον κώδικα «**Controller.prototype = new application.Controller();**» αυτή η κλάση-**Ελεγκτής** επεκτείνει την υπερκλάση «**application.Controller**» για να μπορέσει να έχει πρόσβαση στην μέθοδο δρομολόγησης «**this.route**». Παρακάτω στον κώδικα ορίζονται όλοι οι διαχειριστές συμβάντων, με τους «**editContact**» και «**deleteContact**» να έχουν ως έξτρα όρισμα την μεταβλητή «**id**» της διεύθυνσης **URL.** Όλοι οι διαχειριστές συμβάντων αναλαμβάνουν την ανάθεση καθηκόντων σε **Προβολές** και **Μοντέλα**, ανάλογα με την περίπτωση.

5.1.10 Στατική Έκδοση της πλευράς του Πελάτη

Σε αυτό το σημείο έχουν αναφερθεί και περιγραφτεί όλες οι κύριες κλάσης της πλευράς του Πελάτη. Αξίζει να αναφερθεί το γεγονός ότι υπάρχει δυνατότητα λειτουργίας της πλευράς του Πελάτη αυτόνομα, χωρίς την ανάγκη για ύπαρξη μιας διαδικτυακής υπηρεσίας και πλευράς του Διακομιστή. Αυτή η δυνατότητα δημιουργήθηκε κυρίως για να μπορέσει κάποιος να φορτώσει και να δουλέψει την εφαρμογή, να δει τις λειτουργίες που προσφέρει, χωρίς να χρειαστεί να μπει στην διαδικασία εγκατάστασης ενός **ColdFusion Server** στον υπολογιστή του. Είναι σαν μια «εκτός-σύνδεσης» έκδοση της εφαρμογής. Για να επιτευχθούν όλα τα παραπάνω, αρκεί να αλλαχτεί μία λέξη από τον κώδικα στο κεντρικό αρχείο «**index.html**». Η αλλαγή αυτή γίνεται στις δηλώσεις στην αρχή του κώδικα, και περιλαμβάνει την αλλαγή της δήλωσης για το αρχείο του **Μοντέλου** «**contact_service.js**» σε «**contact_service_static.js**». Το αρχείο «**contact_service_static.js**» είναι μια στατική έκδοση

του **Μοντέλου** «**contact_service.js**». Επειδή όπως αναφέρθηκε και παραπάνω, το μόνο μέρος της πλευράς του Πελάτη που μιλάει με τον Διακομιστή είναι αυτό το **Μοντέλο**, αλλάζοντας αυτό το αρχείο με την στατική έκδοση ουσιαστικά μετατρέπουμε την εφαρμογή σε μια «εκτός-σύνδεσης» εφαρμογή (**offline application**). Η κλάση «**contact_service_static.js**», δεν έχει καμία κλήση **AJAX** μέσα της, δεν χρησιμοποιείται δηλαδή απομακρυσμένη βάση με επαφές σε μορφή **JSON** όπως στην δυναμική της έκδοση, αλλά αντίθετα οι επαφές αποθηκεύονται τοπικά και προσωρινά με χρήση συστοιχιών (**arrays**). Με αυτόν τον τρόπο, μπορεί ο χρήστης να μην αντιλαμβάνεται διαφορά καθώς όλη η λειτουργικότητα τις εφαρμογής είναι παρούσα, όμως δεν υπάρχει βάση με επαφές και δεν υπάρχουν κλήσεις **AJAX** στο παρασκήνιο. Είναι απλά μια έκδοση για όποιον θέλει να δοκιμάσει τις δυνατότητες τις εφαρμογής εύκολα και γρήγορα.

5.2 Πλευρά του Διακομιστή – ColdFusion

Η πλευρά του Διακομιστή σε αυτήν την εφαρμογή αναλαμβάνει την πλευρά της διαδικτυακής υπηρεσίας που χρειάζεται η πλευρά του Πελάτη για να λειτουργήσει. Στην πλευρά του Διακομιστή επίσης βρίσκονται αποθηκευμένα και όλα τα αρχεία της εφαρμογής, καθώς εκεί φιλοξενείται η εφαρμογή, και ο χρήστης θα πρέπει να πληκτρολογήσει την διεύθυνση **URL** της εφαρμογής, που της έχει αποδώσει ο Διακομιστής, για να φορτώσει και να αλληλεπιδράσει με την εφαρμογή. Θα ακολουθήσει ανάλυση μερικών ενδεικτικών τεχνικών θεμάτων της πλευράς του Διακομιστή, η οποία αποτελείται εξ' ολοκλήρου από τεχνολογία **ColdFusion**.

5.2.1 Η χρήση του διαχειριστή συμβάντων «OnCFCRequest()»

Όπως έχει αναφερθεί και στην σχεδίαση της εφαρμογής στην πλευρά του Διακομιστή, το αρχείο «**Application.cfc**» είναι ο πυρήνας της εφαρμογής σε αυτήν την πλευρά. Στο αρχείο «**Application.cfc**» χρησιμοποιείται ο διαχειριστής συμβάντων «**OnCFCRequest()**» για την διαχείριση των **CFC** αιτήσεων. Με την χρήση αυτού του διαχειριστή έχουμε διάφορα οφέλη. Καταρχήν, έχουμε την δυνατότητα να αποθηκεύουμε προσωρινά στην μνήμη (**cache**) τις απομακρυσμένες αιτήσεις για «εξαρτήματα» (**components**) του **ColdFusion**. Σε αντίθετη περίπτωση το **ColdFusion** θα έπρεπε να «συγκεκριμενοποιήσει» (**instantiate**) κάθε «εξάρτημα» που θα ζητούνταν, ενώ τώρα μπορεί να χρησιμοποιήσει μία ήδη έτοιμη «παρουσία» του από την προσωρινή μνήμη. Ένα επιπλέον όφελος είναι η δυνατότητα κατάλληλης διαχείρισης των σφαλμάτων των διεπαφών **APIs** (**APIs Errors**), ενώ ένα πολύ σημαντικό όφελος είναι η δυνατότητα τα δεδομένα της απάντησης να μπορούν να σταλούν σε μία ποικιλία από μορφές, όπως **JSON**, **JSONP**, προσαρμοσμένη **XML** και άλλα.

5.2.2 Γενική Επισκόπηση των κλάσεων

Στην πλευρά του Διακομιστή, όσον αφορά τα **Μοντέλα**, η κλάση «**Contact.cfc**» είναι η κύρια κλάση, αναλαμβάνει της κύριες εργασίες του **Μοντέλου** και λειτουργεί ως η **Πρωταρχική Οντότητα Τομέα** (**Primary Domain Entity**) . Η κλάση «**ContactService.cfc**» διαθέτει τις βασικές μεθόδους που «μιλούν» με την κλάση «**ContactDAO.cfc**» (**DAO: Data Access Object**, αντικείμενο πρόσβασης των δεδομένων) η οποία βοηθάει στην διατήρηση της «διάρκειας» των δεδομένων. Η κλάση «**ContactDAO.cfc**» αναλαμβάνει της «σειριοποίηση» των πληροφοριών (των δεδομένων των επαφών δηλαδή) και την αποθήκευσή τους στο αρχείο «**contacts.json**» το οποίο ουσιαστικά αποτελεί και την μίνι βάση δεδομένων του συστήματος.

Η ουσιαστική μεριά του Διακομιστή που κοιτάζουμε από την οπτική γωνία του Πελάτη, είναι οι διεπαφές **API**. Οι διεπαφές **API** αποτελούνται από τα εξαρτήματα «**Base.cfc**» και «**Contacts.cfc**». Το «εξάρτημα» «**Contacts.cfc**» είναι το εξάρτημα που καλεί ο Πελάτης για να κάνει τις απομακρυσμένες αιτήσεις. Αυτό το εξάρτημα «επεκτείνει» (**extends**) το «εξάρτημα» «**Base.cfc**», το οποίο είναι «εξάρτημα» πυρήνας για όλα τα απομακρυσμένα «εξαρτήματα» **API** που πρόκειται να δημιουργηθούν.

5.2.3 Το «εξάρτημα» πυρήνας «Base.cfc»

```
<cfcomponent
    output="false"
    hint="I provide the base functionality for remote API
components.">
    <cffunction
        name="getFactory"
        access="public"
        returntype="any"
        output="false"
        hint="I return the application factory.">
        <cfreturn application.factory />
    </cffunction>
    <cffunction
        name="getNewResponse"
        access="public"
        returntype="struct"
        output="false"
        hint="I return a new API response object.">
        <cfreturn this.getFactory().newAPIResponse() />
    </cffunction>
</cfcomponent>
```

Το «εξάρτημα» «**Base.cfc**» ουσιαστικά αποτελείται από δύο μεθόδους. Η μία μέθοδος είναι η «**getFactory**», η οποία συλλέγει την «παρουσία» «**factory**» («εργοστάσιο») από την εφαρμογή και την επιστρέφει («**<cfreturn application.factory />**»). Η «**factory**», όπως και στην εφαρμογή της πλευράς του Πελάτη, είναι το «εξάρτημα» στον Διακομιστή το οποίο αναλαμβάνει την δημιουργία των κλάσεων, αναλαμβάνει επίσης τις εγχύσεις εξαρτήσεων,

την αναστροφή του ελέγχου, όπως επίσης αναλαμβάνει και την προσωρινή αποθήκευση στην μνήμη των «**singletons**». Οπότε ουσιαστικά μπορούμε να αποθηκεύσουμε και να ανακτήσουμε «εξαρτήματα» όταν αυτό χρειάζεται, χρησιμοποιώντας αυτήν την «παρουσία» «**factory**».

Η άλλη μέθοδος του εξαρτήματος, είναι η «**getNewResponse**». Η μέθοδος αυτή δημιουργεί μια νέα παρουσία της απάντησης της διεπαφής **API**. Όλες οι απαντήσεις της διεπαφής **API**, έχουν την ίδια δομή. Η δομή αυτή περιλαμβάνει τρία κλειδιά:

- **Success** (τύπου «**boolean**»)
- **Data** (οτιδήποτε τύπου)
- **Errors** (τύπου συστοιχίας ή **array**)

Το κλειδί «**Success**» είναι ουσιαστικά μια «λογική» τιμή για το εάν η αίτηση ήταν επιτυχής ή όχι. Το κλειδί «**Data**» περιέχει την πληροφορία που στέλνεται με την απάντηση, οποιαδήποτε μορφής και αν είναι αυτή (όπως π.χ. μια συμβολοσειρά, ένας αριθμός, μια δομή, μια σειρά από δομές). Το τελευταίο κλειδί «**Errors**» είναι μια συλλογή από σφάλματα, και συγκεκριμένα μια σειρά από σφάλματα σε μια προτυποποιημένη δομή.

5.2.4 Το «εξάρτημα» «Contacts.cfc»

Το «εξάρτημα» «**Contacts.cfc**», του οποίου ο κώδικας δίνεται παρακάτω, αποτελείται από μια σειρά από συναρτήσεις που εκτελούν τις διάφορες λειτουργίες που αιτείται ο Πελάτης. Η πρώτη συνάρτηση, η «**deleteContact**», η οποία αναλαμβάνει την διαγραφή μιας επαφής, δημιουργεί μια νέα «παρουσία» της απάντησης της διεπαφής **API** («**<cfset local.response = this.getNewResponse() />**») και στην συνέχεια καλεί το «εξάρτημα» «**factory**» και του ζητάει το Μοντέλο «**ContactService**» από το οποίο θα καλέσει την μέθοδο «**deleteContact**» και θα της περάσει ως όρισμα το αναγνωριστικό (**id**) της επαφής που πρόκειται να διαγραφεί και το οποίο γνωρίζει από τα δεδομένα της αίτησης («**this.getFactory().getContactService().deleteContact(arguments.id).getAll()**»). Αυτό θα έχει ως αποτέλεσμα την κλήση του στρώματος υπηρεσίας (**service layer**) και την διαγραφή της επαφής. Το στρώμα υπηρεσίας με την σειρά του, επιστρέφει μια «παρουσία» της επαφής που διέγραψε, την οποία η μέθοδος χρησιμοποιεί για να συλλέξει όλες τις ιδιότητες αυτής της επαφής (δηλαδή τα **id**, **name**, **phone**, **email**) και δημιουργεί μια δομή από αυτές, η οποία ορίζεται ως η τιμή του κλειδιού «**Data**» της απάντησης. Στην συνέχεια η μέθοδος θα επιστρέψει όλες τις ιδιότητες της απάντησης του Διακομιστή στον Πελάτη, οι οποίες είναι τα τρία κλειδιά που αναφέρθηκαν και παραπάνω, τα «**Success**», «**Data**» και «**Errors**» («**<cfreturn local.response.getAll() />**»).

```
<cfcomponent
    extends="Base"
    output="false"
    hint="I provide remote API calls for contacts.">
    <cffunction
        name="deleteContact"
        access="remote"
        returntype="struct"
        returnformat="json"
        output="false"
        hint="I delete the contact at the given ID.">
        <cfargument
            name="id"
            type="numeric"
            required="true"
            hint="I am the ID of the contact being deleted."
            />
        <cfset var local = {} />
        <cfset local.response = this.getNewResponse() />
        <cfset local.response.setData(
            this.getFactory().getContactService().deleteContact(
arguments.id ).getAll()
            ) />
        <cfreturn local.response.getAll() />
    </cffunction>
    <cffunction
        name="getContact"
        access="remote"
        returntype="struct"
        returnformat="json"
        output="false"
        hint="I return the contact at the given ID.">
        <cfargument
            name="id"
            type="numeric"
            required="true"
            hint="I am the ID of the contact being gotten."
            />
        <cfset var local = {} />
        <cfset local.response = this.getNewResponse() />
        <cfset local.response.setData(
            this.getFactory().getContactService().getContact(
arguments.id ).getAll()
            ) />
        <cfreturn local.response.getAll() />
    </cffunction>
    <cffunction
        name="getContacts"
        access="remote"
        returntype="struct"
        returnformat="json"
        output="false"
        hint="I return the contacts.">
        <cfset var local = {} />
        <cfset local.response = this.getNewResponse() />
        <cfset local.contacts = [] />
        <cfloop
```

```
            index="local.contact"

array="#this.getFactory().getContactService().getContacts()#">
            <cfset arrayAppend(
                local.contacts,
                local.contact.getAll()
                ) />
        </cfloop>
        <cfset local.response.setData( local.contacts ) />
        <cfreturn local.response.getAll() />
    </cffunction>
    <cffunction
        name="saveContact"
        access="remote"
        returntype="struct"
        returnformat="json"
        output="false"
        hint="I save the given contact data. If there are any
valiation errors, they will be returned in the response.">
        <cfargument
            name="id"
            type="numeric"
            required="true"
            hint="I am the ID of the contact."
            />
        <cfargument
            name="name"
            type="string"
            required="true"
            hint="I am the name of the contact."
            />
        <cfargument
            name="phone"
            type="string"
            required="true"
            hint="I am the phone number of the contact."
            />
        <cfargument
            name="email"
            type="string"
            required="true"
            hint="I am the email address of the contact."
            />
        <cfset var local = {} />
        <cfset local.response = this.getNewResponse() />
        <cfset local.contact =
this.getFactory().getContactService().getContact( val(
arguments.id ) ) />
        <cfset local.contact.setAll(
            name = arguments.name,
            phone = arguments.phone,
            email = arguments.email
            ) />
        <cfset local.errors = local.contact.validate() />
        <cfif arrayLen( local.errors )>
            <cfset local.response.applyValidationErrors(
local.errors ) />
        <cfelse>
```

```
<cfset this.getFactory().getContactService().saveContact(
local.contact ) />
            <cfset local.response.setData( local.contact.getId() )
/>
        </cfif>
        <cfreturn local.response.getAll() />
    </cffunction>
</cfcomponent>
```

Ο υπόλοιπος κώδικας λειτουργεί ακριβώς με τον ίδιο τρόπο, όλες οι συναρτήσεις δημιουργούν μια νέα «παρουσία» της απάντησης, στην συνέχεια καλούν μέσω της μεθόδου «**getFactory**» την κατάλληλη μέθοδο για την ενέργεια που έχει ζητηθεί από τον Πελάτη, η ενέργεια εκτελείται από το στρώμα υπηρεσίας το οποίο γυρίζει τις ιδιότητες οι οποίες χρησιμοποιούνται για την επιστροφή της απάντησης στον Πελάτη.

Όλες οι μέθοδοι έχουν τύπο πρόσβασης «**remote**» («**access="remote"**») δηλαδή «απομακρυσμένου» καθώς θα εκτελούνται απομακρυσμένα από την εφαρμογή και όλες έχουν τύπο επιστροφής «**struct**» («**returntype="struct"**») καθώς όλες επιστρέφουν αυτές τις δομές απάντησης που είδαμε προηγουμένως. Επιπλέον, η μορφή των δεδομένων που επιστρέφονται είναι φυσικά το **JSON** («**returnformat="json"**»), το οποίο είναι η προεπιλογή, όμως δεν είναι απαραίτητο να χρησιμοποιηθεί καθώς η εφαρμογή μας επιτρέπει να επιλέξουμε και άλλη μορφή δεδομένων, όπως π.χ. απλό κείμενο, και αλλάζοντας τον τύπο της μορφής το **ColdFusion** θα μετατρέψει αυτόματα τα δεδομένα στην μορφή που ζητήθηκε και θα τα επιστρέψει με την απάντηση. Παρόλα αυτά, το **ColdFusion** θα επιστρέψει τα δεδομένα με τα κλειδιά της δομής γραμμένα στα κεφαλαία, και αυτός είναι ο λόγος που από την πλευρά του Πελάτη έχει ρυθμιστεί η εφαρμογή να «ομαλοποιεί» (δηλαδή να μετατρέπει σε πεζά γράμματα) τα δεδομένα της απάντησης που δέχεται.

Αυτή ήταν η ανάλυση ενός ενδεικτικού μέρους του κώδικα στον Διακομιστή. Σε αυτό το σημείο θα πρέπει να επισημανθεί ότι όλος ο κώδικας της εφαρμογής (και των δυο πλευρών) έχει συνταχθεί περιέχοντας αναλυτικά σχόλια για τα μέρη που χωρίζεται και το τι ενέργειες εκτελούν αυτά. Ο αναγνώστης θα μπορέσει διαβάζοντας τα σχόλια να κατανοήσει πλήρως τις περιγραφόμενες λειτουργίες του κώδικα.

5.3 Πλατφόρμες και προγραμματιστικά εργαλεία

Για την ανάπτυξη της εφαρμογής «**Έξυπνες Επαφές**» της πτυχιακής εργασίας χρησιμοποιήθηκαν τα παρακάτω προγραμματιστικά εργαλεία και πλατφόρμες. Για την πλευρά του Διακομιστή χρησιμοποιήθηκε η πλατφόρμα **ColdFusion 9** της εταιρίας **Adobe Systems**, η οποία προσφέρει ολοκληρωμένη τεχνολογία για την πλευρά αυτή, δηλαδή και

τεχνολογία για την εγκατάσταση του Διακομιστή αυτού καθεαυτού αλλά και για την ανάπτυξη της τεχνολογίας της διαδικτυακής υπηρεσίας-εφαρμογής που θα εκτελείται. Η πλατφόρμα **ColdFusion** είναι ένα εμπορικό προϊόν όμως υπάρχει η δυνατότητα «κατεβάσματος» και εγκατάστασης της έκδοσης «**Developer Edition**», η οποία διατίθεται δωρεάν με έναν περιορισμό. Η έκδοση αυτή είναι μια πλήρης έκδοση, εφάμιλλη της εμπορικής, που προορίζεται για την ανάπτυξη εφαρμογών τοπικά από τους προγραμματιστές. Ο μόνος περιορισμός σε σχέση με την βασική εμπορική έκδοση είναι ότι η πρόσβαση στις εφαρμογές που τρέχουν σε Διακομιστή της «**Developer Edition**» έχει περιοριστεί σε δύο μηχανήματα Πελάτες. Η έκδοση προσφέρεται για κατέβασμα στην διεύθυνση: http://www.adobe.com/cfusion/tdrc/index.cfm?product=coldfusion .

Για την συγγραφή του κώδικα στην πλευρά του Διακομιστή, δηλαδή των αρχείων «**.cfc**» και «**.cfm**», χρησιμοποιήθηκε το πρόγραμμα **Adobe ColdFusion Builder**, κυρίως για την δυνατότητα του χρωματισμού (**syntax highlighting**) στην σύνταξη του κώδικα της γλώσσας **CFML**. Την περίοδο ανάπτυξης της πτυχιακής εργασίας και κατά συνέπεια και της εφαρμογής, το **ColdFusion Builder** προσφερόταν δωρεάν καθώς ήταν σε δοκιμαστική φάση ανάπτυξης (**beta**). Με τον ερχομό της τελικής έκδοσης το πρόγραμμα πέρασε σε εμπορική διάθεση, και προσφέρεται σε δοκιμαστική πλήρη έκδοση εξήντα ημερών στην διεύθυνση: http://www.adobe.com/cfusion/tdrc/index.cfm?product=coldfusion_builder . Να σημειωθεί ότι το πρόγραμμα δεν απαιτείται για την λειτουργία ή την επεξεργασία του κώδικα της εφαρμογής. Για την επεξεργασία του κώδικα των προαναφερθέντων αρχείων μπορεί να χρησιμοποιηθεί οποιοδήποτε πρόγραμμα ανάγνωσης κώδικα, όπως το παρακάτω.

Για την συγγραφή του κώδικα των τεχνολογιών στην πλευρά του πελάτη (**JavaScript**, **CSS**, **HTML**) καθώς και την επεξεργασία του αρχείου «**contacts.json**» χρησιμοποιήθηκε το δωρεάν πρόγραμμα **NotePad++**. Το συγκεκριμένο πρόγραμμα ανήκει στην κατηγορία του λογισμικού ανοικτού κώδικα (**open source**) με μια πολύ ενεργή κοινότητα και πλούσια χαρακτηριστικά. Πρόκειται για έναν επεξεργαστή κειμένου με δυνατότητες χρωματισμού της σύνταξης του κώδικα πολλών γλωσσών προγραμματισμού, και την δυνατότητα ενσωμάτωσης δωρεάν επιπρόσθετων προγραμμάτων (**plug-ins**) για έξτρα λειτουργίες. Επιλέχτηκε κυρίως για την απλότητα και τις πολύ μικρές απαιτήσεις του σε υπολογιστική ισχύ.

Τέλος, για την κατασκευή των εικόνων και σχημάτων του κειμένου αυτού, καθώς και για την κατασκευή του λογότυπου της εφαρμογής, χρησιμοποιήθηκε το πρόγραμμα **Fireworks CS4** της **Adobe Systems**, το οποίο προσφέρεται στην διεύθυνση: http://www.adobe.com/cfusion/tdrc/index.cfm?product=fireworks σε δοκιμαστική πλήρη έκδοση τριάντα ημερών. Το πρόγραμμα επιλέχτηκε κυρίως για την ευκολία που προσφέρει στην παραγωγή γραφικών για χρήση στο διαδίκτυο. Για την κατασκευή του φόντου της

εφαρμογής, χρησιμοποιήθηκε η διαδικτυακή εφαρμογή **Stripe Generator 2.0**, η οποία προσφέρεται δωρεάν και φιλοξενείται στην διεύθυνση: http://www.stripegenerator.com .

Κατά την διάρκεια δοκιμών και αποσφαλμάτωσης της εφαρμογή, χρησιμοποιήθηκε το πρόγραμμα περιήγησης **Firefox 3.5** (και αργότερα **3.6**) με εγκατεστημένα τα πρόσθετα προγράμματα (**Add-ons**) **Web Developer** και **Firebug**. Τα παραπάνω χρησιμοποιήθηκαν για τις προηγμένες δυνατότητες που προσέφεραν στην παρακολούθηση του κώδικα της **JavaScript** και την πλήρη καταγραφή και εμφάνιση των σφαλμάτων και των προειδοποιητικών μηνυμάτων που δημιουργούνται κατά την εκτέλεση της εφαρμογής.

5.3.1 Διαδικασία εγκατάστασης της πτυχιακής εργασίας σε υπολογιστή

Για την εγκατάσταση και την λειτουργία την εφαρμογής «**Έξυπνες Επαφές**» της πτυχιακής εργασίας, χρειάζονται δυο βασικά βήματα, το πρώτο αφορά την εγκατάσταση της πλατφόρμας **ColdFusion** και τη ρύθμιση του Διακομιστή της, ενώ το δεύτερο αφορά την αντιγραφή του φάκελου «**application**» από το **CD** της πτυχιακής στον φάκελο «**wwwroot**» του Διακομιστή. Αναλυτικά τα βήματα παρουσιάζονται παρακάτω. Να σημειωθεί ότι η οδηγίες αυτές αφορούν την εγκατάσταση σε λειτουργικό περιβάλλον **Windows** της **Microsoft**.

5.3.1.1 Εγκατάσταση του ColdFusion

Για την εγκατάσταση του Διακομιστή θα χρησιμοποιηθεί η έκδοση «**Developer Edition**» του **ColdFusion** η οποία μπορεί να κατεβαστεί από την διεύθυνση που αναφέρεται παραπάνω αλλά περιέχεται και στον φάκελο «**coldfusion**» στο **CD** της πτυχιακής.

Για την εγκατάσταση από το **CD** της πτυχιακής, απλά χρειάζεται η εκτέλεση του αρχείου «**ColdFusion_9_WWE_win.exe**» που βρίσκονται εντός του προαναφερθέντα φάκελου. Κατά την διάρκεια της εγκατάστασης θα χρειαστεί να επιλεγούν διάφορες ρυθμίσεις. Αρχικά στην οθόνη «**Install Type**» θα πρέπει να επιλεγεί η επιλογή «**Developer Edition**» καθώς είναι η έκδοση που πρόκειται να χρησιμοποιηθεί. Στην επόμενη οθόνη με τίτλο «**Installer Configuration**» πρέπει να επιλεγεί η ρύθμιση «**Server configuration**». Στην συνέχεια, στην οθόνη «**Subcomponent Installation**» αφήνουμε όλες τις προεπιλεγμένες επιλογές, ενώ στην επόμενη οθόνη με τίτλο «**Select Installation Directory**» δίνεται η δυνατότητα επιλογής του φακέλου εγκατάστασης της πλατφόρμας του Διακομιστή. Η επιλογή είναι ελεύθερη, αλλά για λόγους ευκολίας η προεπιλεγμένη ρύθμιση «**C:\ColdFusion9**» θα χρησιμοποιηθεί για αυτές τις οδηγίες. Στην επόμενη οθόνη με τίτλο «**Configure Web Servers/Websites**» επιλέγουμε την επιλογή «**Build-in web server**», ενώ στην επόμενη οθόνη με τίτλο «**Administrator Password**» θα πρέπει να πληκτρολογηθεί ένας κωδικός διαχειριστή δυο φορές. Στην

τελευταία οθόνη πριν από την εγκατάσταση, με τίτλο «**Enable RDS**» αφήνουμε τις προεπιλεγμένες ρυθμίσεις και στην συνέχεια πραγματοποιείται η εγκατάσταση.

5.3.1.2 Ρύθμιση του ColdFusion Server

Μετά την ολοκλήρωση της εγκατάστασης, θα χρειαστεί να γίνουν κάποιες ρυθμίσεις στον Διακομιστή. Στο τέλος της εγκατάστασης, θα προσφερθεί η επιλογή «**Launch the Configuration Wizard in the default browser**» την οποία αφήνουμε επιλεγμένη και πατάμε το κουμπί «**Done**». Με αυτήν την επιλογή, η εγκατάσταση θα τερματιστεί και θα εκκινήσει το προεπιλεγμένο πρόγραμμα περιήγησης του συστήματος στο οποίο θα έχει φορτωθεί ένας οδηγός ρυθμίσεων για τον Διακομιστή **ColdFusion**.Σε περίπτωση που δεν πραγματοποιηθεί το παραπάνω βήμα για κάποιο λόγο, η διεύθυνση αυτού του οδηγού είναι η: **http://127.0.0.1:8500/CFIDE/administrator/index.cfm**. Στην πρώτη οθόνη που εμφανίζεται πληκτρολογούμε το κωδικό διαχειριστή που έχουμε δηλώσει στην εγκατάσταση και πατάμε το κουμπί «**Login**». Στην συνέχεια γίνεται μια σειρά από αυτόματες αρχικές ρυθμίσεις στον Διακομιστή κατά της οποίες θα χρειαστεί να περιμένουμε. Μετά από αυτές τις ρυθμίσεις, θα εμφανιστεί μια σελίδα που θα αναγράφει «**Setup Complete**» δηλαδή ότι η εγκατάσταση του Διακομιστή έχει πλέον ολοκληρωθεί. Πατώντας το κουμπί «**OK**» θα πρέπει να μεταφερθούμε στην σελίδα «**Adobe ColdFusion Administrator**» που είναι η σελίδα του διαχειριστή του Διακομιστή, και εφόσον γίνει αυτό, σημαίνει ότι ο Διακομιστής έχει εγκατασταθεί σωστά και έχει εκκινήσει με επιτυχία.

5.3.1.3 Αντιγραφή των αρχείων της εφαρμογής και εκτέλεση της εφαρμογής

Στην συνέχεια θα πρέπει να γίνει αντιγραφή ολόκληρου του φακέλου «**application**» ο οποίος περιέχει τα αρχεία της εφαρμογής, και επικόλλησή του μέσα στον φάκελο «**wwwroot**» του Διακομιστή, ο οποίος βρίσκεται στον κατάλογο εγκατάστασης του **ColdFusion**. Εάν ακολουθήθηκαν οι οδηγίες που δόθηκαν παραπάνω, ο φάκελος αυτός θα βρίσκεται στην διαδρομή: «**C:\ColdFusion9\wwwroot**». Τέλος για να εκτελεστεί η εφαρμογή, αρκεί να εκκινήσουμε το πρόγραμμα περιήγησης του συστήματός μας και να πληκτρολογήσουμε την διεύθυνση: «**http://localhost:8500/application/**». Αυτό αφορά την περίπτωση τοπικής εκτέλεσης της εφαρμογής, ενώ σε περίπτωση που η εφαρμογή εγκατασταθεί σε υπολογιστή-Διακομιστή και η πρόσβαση γίνεται απομακρυσμένα, η διεύθυνση παραμένει η ίδια με την διαφορά ότι θα πρέπει αντί του «**localhost**» να πληκτρολογηθεί η διεύθυνση του Διακομιστή στο διαδίκτυο (π.χ. **www.myserver.gr**). Σε αυτήν την περίπτωση η διεύθυνση είναι της μορφής: «**http://www.myserver.gr:8500/application/**».

6

Έλεγχος

Σε αυτήν την ενότητα θα γίνει η αξιολόγηση του συστήματος. Θα ελεγχτεί ότι η εφαρμογή δουλεύει σωστά και ότι όλες οι λειτουργίες που θεωρητικά προσφέρει λειτουργούν όπως ορίζονται.

6.1 Μεθοδολογία ελέγχου

Η αξιολόγηση θα γίνει με την βοήθεια ενός σεναρίου λειτουργίας. Ακολουθώντας το σενάριο ο αναγνώστης θα μπορέσει να γνωρίσει και πως μπορεί να χρησιμοποιήσει τις λειτουργίες της εφαρμογής. Το σενάριο περιλαμβάνει τα εξής: Ο χρήστης αρχικά πηγαίνει στην διεύθυνση της εφαρμογής και φορτώνεται η κεντρική σελίδα της. Ο χρήστης διατρέχει την λίστα με τις επαφές του και μεταφέρεται στην κορυφή της σελίδας με την χρήση του κατάλληλου κουμπιού. Στην συνέχεια κάνει μια αναζήτηση στις επαφές του για να βρει και να εμφανίσει μερικές που τον ενδιαφέρουν και επεξεργάζεται τα στοιχεία μίας εξ' αυτών. Στην συνέχεια αναζητεί μια επαφή που δεν υπάρχει και την οποία έπειτα δημιουργεί. Ο χρήστης αναζητεί και βρίσκει την επαφή που δημιούργησε προηγουμένως, την οποία και διαγράφει ενώ στην συνέχεια την αναζητεί ξανά για να ελέγξει ότι δεν υπάρχει πια. Τέλος, ο χρήστης προσπαθεί να δημιουργήσει μια νέα επαφή χωρίς να δώσει όνομα και η εφαρμογή τον ειδοποιεί αντίστοιχα.

6.2 *Αναλυτική παρουσίαση ελέγχου*

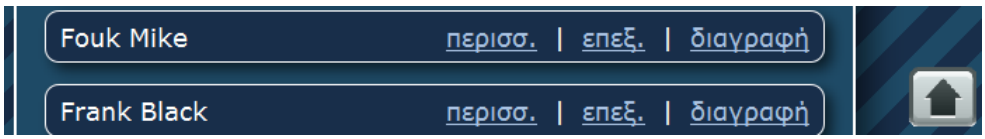
Το σενάριο ξεκινάει με τον χρήστη να πληκτρολογεί την διεύθυνση της εφαρμογής και στην συνέχεια η εφαρμογή να φορτώνεται και να εμφανίζει τα περιεχόμενα της κεντρικής της σελίδας. Στην περίπτωσή μας η διεύθυνση της εφαρμογής είναι η http://localhost:8500/application/ εφόσον αποφασίσουμε να δοκιμάσουμε την εφαρμογή τοπικά, από τον Διακομιστή που έχουμε εγκαταστήσει σύμφωνα με της οδηγίες που δόθηκαν σε προηγούμενη ενότητα.

6.2.1 Φόρτωση και Αρχική Σελίδα

Η κεντρική σελίδα της εφαρμογή φορτώνει και εμφανίζεται με την μορφή που φαίνεται στην εικόνα που ακολουθεί:

Στην κεντρική σελίδα φαίνονται, κάτω από το λογότυπο της εφαρμογής, το πεδίο αναζήτησης όπου ο χρήστης πληκτρολογεί το αντικείμενο της αναζήτησής του. Ο χρήστης μπορεί να αναζητήσει μια επαφή, πληκτρολογώντας οποιοδήποτε από τα στοιχεία της που πιθανό να γνωρίζει, δηλαδή το όνομα της επαφής, τον αριθμό τηλεφώνου και την διεύθυνση ηλεκτρονικού ταχυδρομείου. Αρχικά ο χρήστης εκτελεί μια απλή επισκόπηση της λίστας των επαφών του, κυλώντας τα περιεχόμενα τις σελίδας προς τα κάτω, ενώ όταν θελήσει να

ξαναέρθει στην κορυφή της σελίδας μπορεί να το κάνει πατώντας το κουμπί που εμφανίζεται στην κάτω δεξιά γωνία της σελίδας, το οποίο τον μεταφέρει αυτόματα στην κορυφή με την βοήθεια ενός εφέ κύλισης προς τα πάνω. Το κουμπί αυτό φαίνεται στην παρακάτω εικόνα:



6.2.2 Αναζήτηση επαφής

Σύμφωνα με το σενάριο, ο χρήστης αποφασίζει να αναζητήσει μια επαφή με το όνομα «**maria**», και επομένως πληκτρολογεί τα αντίστοιχα γράμματα. Από την πληκτρολόγηση του πρώτου γράμματος, ο χρήστης παίρνει μια πρώτη ανταπόκριση από την εφαρμογή, καθώς ήδη αρχίζει να φιλτράρεται η λίστα με τις επαφές που εμφανίζονται, στην οποία απομένουν μόνο οι επαφές που περιέχουν τα πληκτρολογημένα γράμματα κάθε δεδομένη στιγμή. Με την ολοκλήρωση της πληκτρολόγησης του ονόματος «**maria**» ο χρήστης βλέπει μπροστά του την ακόλουθη οθόνη:

Ο χρήστης διαπιστώνει ότι έχει καταχωρήσει 3 επαφές που το όνομά τους περιέχει το «**maria**», και μπορεί πλέον να επικεντρωθεί στην συγκεκριμένη επαφή που έψαχνε. Έχει την δυνατότητα να δει περισσότερες πληροφορίες για την επαφή που τον ενδιαφέρει, πατώντας πάνω στα στην λέξη «**περισσ.**» που υπάρχει δίπλα από το όνομα της επαφής. Εναλλακτικά, και εφόσον η χρήστης επιθυμεί να δει επιπλέον στοιχεία και για τις τρεις επαφές που εμφανίστηκαν, μπορεί να πατήσει το κουμπί «**Enter**» του πληκτρολογίου του και με ένα εφέ αναδίπλωσης προς τα κάτω, θα εμφανιστούν τα στοιχεία των τριών πρώτων επαφών της λίστας. Αυτή η δυνατότητα παρέχεται ανά πάσα στιγμή στον χρήστη, ακόμη και αμέσως μετά

την αρχική φόρτωση της εφαρμογής. Στην εικόνα που ακολουθεί φαίνονται οι επαφές όπως είναι μετά την εμφάνιση των πλήρων στοιχείων τους:

6.2.3 Επεξεργασία Επαφής

Στην συνέχεια ο χρήστης αποφασίζει ότι η επαφή που έψαχνε, με το όνομα «**Maria Goldburg**», έχει κάποιο λάθος στοιχείο και θέλει να το διορθώσει. Πατώντας λοιπόν στο κουμπί «**επεξ.**» δίπλα στο όνομα αυτής της επαφής, μεταφέρεται στην φόρμα επεξεργασίας επαφής, η οποία φαίνεται παραπάνω.

Από την οθόνη αυτή, και αλλάζοντας τα δεδομένα που επιθυμεί στο κατάλληλο πεδίο, μπορεί να κάνει μετατροπές στα στοιχεία της επαφής. Πατώντας το κουμπί «**Αποθήκευση & Κλείσιμο**» οι όποιες αλλαγές που έχει κάνει αποθηκεύονται, ενώ πατώντας στο κουμπί «**Ακύρωση**» δεν αποθηκεύεται καμία αλλαγή και ο χρήστης λαμβάνει ένα μήνυμα επιβεβαίωσης για την ενδεχόμενη ακύρωση. Και στις δυο περιπτώσεις όμως, όποιο κουμπί και αν πατήσει, ο χρήστης τελικά μεταφέρεται πίσω στην κεντρική σελίδα, με τις ενδεχόμενες αλλαγές που έχει κάνει να έχουν (ή να μην έχουν, αντίστοιχα) ενσωματωθεί στην λίστα. Επίσης, ο χρήστης μπορεί να πατήσει και πάνω στην λέξη «**πίσω**» που φαίνεται στην κορυφή της φόρμας και να γυρίσει πίσω στην κεντρική σελίδα, χωρίς να αποθηκευτεί καμία αλλαγή, αλλά και χωρίς κανένα μήνυμα επιβεβαίωσης.

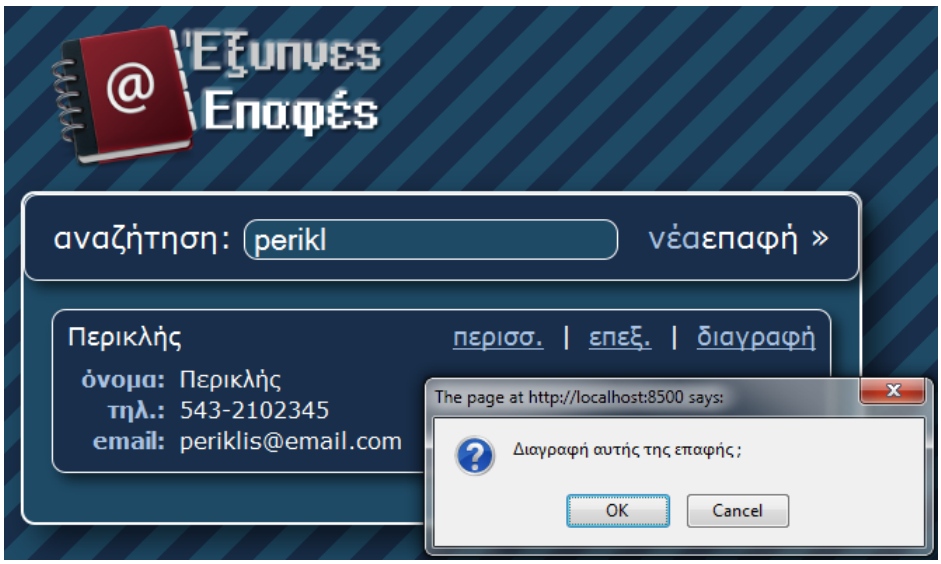
6.2.4 Δημιουργία νέας επαφής

Σύμφωνα με το σενάριο, ο χρήστης ενώ βρίσκεται στην κεντρική σελίδα αποφασίζει να αναζητήσει την επαφή με το όνομα «**Περικλής**». Όταν πληκτρολογεί το όνομα αυτό, δεν εμφανίζεται κανένα αποτέλεσμα, και ο χρήστης αποφασίζει να δημιουργήσει αυτήν την επαφή. Αυτό μπορεί να το κάνει με δυο τρόπους, πρώτον, πατώντας απλά το κουμπί «**Enter**» του πληκτρολογίου του ενώ βρίσκεται στο πεδίο αναζήτησης, και δεύτερον πατώντας με το ποντίκι του στις λέξεις «**νέαεπαφή**» που βρίσκονται δίπλα από το πεδίο αναζήτησης. Και στις δύο περιπτώσεις ο χρήστης θα μεταφερθεί στην φόρμα δημιουργίας επαφής, η οποία εμφανίζεται έχοντας ήδη «γεμίσει» το πεδίο του ονόματος με τα περιεχόμενα του πεδίου αναζήτησης (εφόσον υπάρχουν), δηλαδή την λέξη «**Περικλής**», όπως φαίνεται και στην παρακάτω εικόνα.

Το «εφόσον υπάρχουν» αναφέρεται, καθώς υπάρχει και η δυνατότητα ο χρήστης να μεταφερθεί στην φόρμα νέα επαφής ανά πάσα στιγμή, πατώντας απλά τις λέξεις «**νέαεπαφή**» από την κεντρική σελίδα, χωρίς να έχει πληκτρολογήσει κάτι στο πεδίο αναζήτησης, και επομένως το πεδίο του ονόματος της φόρμας θα εμφανιστεί κενό.

Στην συνέχεια ο χρήστης πληκτρολογεί τα στοιχεία της επαφής, και καθώς αυτή η οθόνη του είναι γνώριμη από την φόρμα επεξεργασίας επαφής, μπορεί να πατήσει πάνω στις 3 επιλογές («**πίσω**», «**Αποθήκευση & Κλείσιμο**» και «**Ακύρωση**») οι οποίες εκτελούν τις ίδιες λειτουργίες με αυτές της φόρμας επεξεργασίας. Ο χρήστης γεμίζει τα πεδία με τα στοιχεία τις επαφής και αποθηκεύει τις αλλαγές πατώντας το κουμπί «**Αποθήκευση & Κλείσιμο**», γυρνώντας στην συνέχεια αυτόματα στην κεντρική σελίδα.

6.2.5 Διαγραφή Επαφής



Βρισκόμενος στην κεντρική σελίδα, ο χρήστης μετανιώνει για την δημιουργία της προηγούμενης επαφής και αποφασίζει να την εντοπίσει και να την διαγράψει. Αποφασίζει να την αναζητήσει χρησιμοποιώντας την διεύθυνση ηλεκτρονικού ταχυδρομείου τις επαφής

(**periklis@email.com**), καθώς ήταν η τελευταία που έγραψε και την θυμάται πιο έντονα. Στο πεδίο αναζήτησης πληκτρολογεί τα γράμματα «**perikl**», και βλέπει ότι η λίστα πλέον περιέχει μόνο την επαφή που θέλει να διαγράψει. Πατώντας το κουμπί «**Enter**» του πληκτρολογίου του, εμφανίζει τα στοιχεία τις επαφής για να βεβαιωθεί ότι είναι η συγκεκριμένη που θέλει να διαγράψει, και στην συνέχεια πατάει με το ποντίκι του πάνω στην λέξη «**διαγραφή**» που βρίσκεται δίπλα στο όνομα της επαφής και στο τέλος της σειράς. Η εφαρμογή εμφανίζει ένα «παράθυρο επιβεβαίωσης» (**alert box**) για την επιβεβαίωση της διαγραφής, και πατώντας το κουμπί «**OK**» ο χρήστης διαγράφει την επαφή και η εφαρμογή τον γυρίζει πίσω στην αρχική σελίδα, με την ανανεωμένη πλέον λίστα επαφών του.

Ο χρήστης για να ελέγξει ότι διαγράφτηκε η επαφή, πληκτρολογεί το όνομα «**Περικλής**» στο πεδίο αναζήτησης και διαπιστώνει ότι καμία επαφή δεν εμφανίζεται σαν αποτέλεσμα στην λίστα.

6.2.6 Μη συμπλήρωση του πεδίου του ονόματος

Κατά την τελευταία πράξη του σεναρίου λειτουργίας, ο χρήστης κάνει κλικ με το ποντίκι του στην σύνδεση «**νέαεπαφή**» και κατά την διάρκεια της δημιουργίας της νέας επαφής, δεν συμπληρώνει τίποτα στο πεδίο της φόρμας που αντιστοιχεί στο όνομα της επαφής. Πατώντας το κουμπί «**Αποθήκευση & Κλείσιμο**», ο χρήστης ειδοποιείται από την εφαρμογή ότι πρέπει να συμπληρώσει το πεδίο ονόματος με το αντίστοιχο μήνυμα που φαίνεται στην παραπάνω εικόνα.

Στην συνέχεια ο χρήστης πληκτρολογεί στο πεδίο του ονόματος το επιθυμητό όνομα της νέας επαφής και αποθηκεύει με επιτυχία την επαφή. Τέλος, ο χρήστης αναζητά την επαφή, πληκτρολογώντας το όνομα της στο πεδίο αναζήτησης ώσπου στην λίστα επαφών εμφανίζεται μόνο η επαφή που δημιούργησε τελευταία. Πατώντας τότε τον συνδυασμό των κουμπιών «**ALT+Enter**» του πληκτρολογίου του, εμφανίζεται το παράθυρο επιβεβαίωσης διαγραφής, όπου ο χρήστης επιλέγει τελικά να την διαγράψει. Εδώ τελειώνει και το σενάριο λειτουργίας για τον έλεγχο της εφαρμογής.

7

Επίλογος

7.1 Σύνοψη και συμπεράσματα

Είναι σημαντικό να τονιστεί ότι η εφαρμογή αυτής της πτυχιακής εργασίας, δεν κατασκευάστηκε με το σκεπτικό να γίνει μια πλήρης πολυτελή εφαρμογή, η οποία θα δουλεύει απρόσκοπτα και θα παρέχει απίστευτες δυνατότητες. Ο σκοπός ήταν να γίνει μια παρουσίαση και μια χειροπιαστή εφαρμογή της τεχνολογίας **AJAX** και όλων των αλληλένδετων τεχνολογιών που την απαρτίζουν, έτσι ώστε να κατανοηθούν και να παρουσιαστούν στην πράξη όσα πλεονεκτήματα προσφέρει για τις διαδικτυακές εφαρμογές και τις μελλοντικές υλοποιήσεις τους. Ο απώτερος στόχος της πτυχιακής δεν ήταν να δοθεί μια μόνιμη και πανάκεια λύση για όλα τα προβλήματα των διαδικτυακών υπηρεσιών και εφαρμογών αλλά να δοθούν κάποιες νέες ιδέες για τον εμπλουτισμό της λειτουργικότητάς τους και της ευχρηστίας που προσφέρουν στον χρήστη. Επίσης ένας άλλος στόχος αυτής της εργασίας ήταν η απομυθοποίηση της πολυπλοκότητας στην πλευρά του Πελάτη. Η ανάπτυξη εφαρμογών που προσανατολίζονται στην πλευρά του πελάτη θεωρείται αρκετά πολύπλοκη κάτι που στην πράξη αποδεικνύεται λάθος. Εάν δεχτούμε την φιλοσοφία που αναπτύσσεται για πλουσιότερες και χρηστικότερες εφαρμογές, θα πρέπει να αρχίσουμε να υιοθετούμε και την τάση που προτείνει την μεταφορά του φόρτου και των λειτουργιών μιας εφαρμογής στην πλευρά του Πελάτη. Με τα σημερινά επίπεδα ποιότητας και ταχύτητας στην τεχνολογία, ο Πελάτης είναι πλέον μια αρκετά στιβαρή επένδυση για το μέλλον στην ανάπτυξη πλούσιων διαδικτυακών εφαρμογών.

7.2 Μελλοντικές επεκτάσεις

Όσον αφορά την μελλοντική επέκταση της εφαρμογής, θα μπορούσαν να ενσωματωθούν αρκετές ιδέες έτσι ώστε η εφαρμογή να γίνει μια πιο πλήρης διαδικτυακή εφαρμογή. Μια από αυτές είναι η δυνατότητα διατήρησης βάσης δεδομένων για τα δεδομένα των επαφών, οι οποίες θα είναι οργανωμένες και ανά χρήστες, καθώς το σύστημα θα περιλαμβάνει σύστημα κρυπτογραφημένης πρόσβασης, και οι χρήστες θα χρησιμοποιούν ένα «όνομα χρήστη» και έναν «κωδικό πρόσβασης» που θα τους έχει δοθεί για να έχουν πρόσβαση σε μια διαδικτυακή βάση με τις επαφές τους. Έτσι οι χρήστες θα μπορούν να διατηρούν ένα διαδικτυακό κατάλογο με τις επαφές τους και να έχουν πρόσβαση σε αυτόν χρησιμοποιώντας τον λογαριασμό τους από οποιοδήποτε σημείο επιθυμούν.

Επίσης, η εφαρμογή θα μπορούσε να εμπλουτιστεί με την προσθήκη επιπλέον στοιχείων για κάθε επαφή, όπως δευτερεύοντα τηλέφωνα, διευθύνσεις κατοικίας και γραφείου, ημερομηνίες γενεθλίων, πεδίο για σημειώσεις και άλλα. Με αυτά τα στοιχεία η εφαρμογή θα μπορούσε να ενσωματώσει και άλλες λειτουργίες όπως η ειδοποιήσεις για όσες ημερομηνίες γενεθλίων των επαφών πλησιάζουν, προσθήκη και διαχείριση λίστας με εργασίες (task-list) τις οποίες ο χρήστης θα διατηρεί και θα λαμβάνει ενημερώσεις, κτλ. Από αισθητικής άποψης, θα μπορούσε να ενσωματωθεί και μια επιλογή η οποία θα επέτρεπε στον χρήστη να επιλέγει πιο φύλλο στυλ (CSS) ταιριάζει με τα γούστα του και αναλόγως η εμφάνιση της εφαρμογής θα προσαρμοζόταν, αλλάζοντας χρώματα, φόντο, γραμματοσειρές και άλλα.

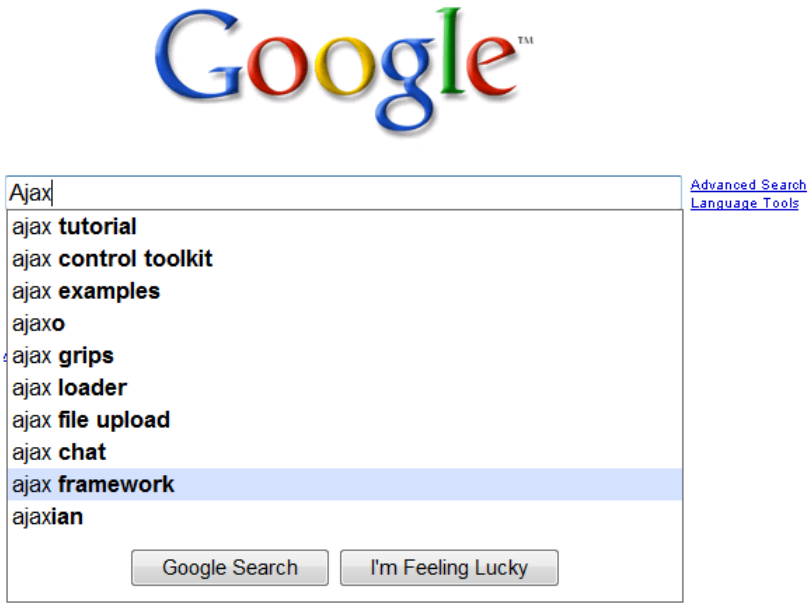
8

Παράρτημα: Σχετικές εφαρμογές με χρήση AJAX

Σήμερα, σχεδόν πέντε χρόνια μετά την πρώτη εφαρμογή της τεχνολογίας **AJAX**, υπάρχουν πολλά και πετυχημένα παραδείγματα, στα οποία η εφαρμογή της τεχνολογίας βοήθησε στην λειτουργικότητα και αποδοτικότητα των διαδικτυακών εφαρμογών. Η πρώτη εταιρία που ασχολήθηκε σοβαρά με το **AJAX**, η **Google**, είναι και αυτή που μέχρι σήμερα κάνει την πιο εκτενή χρήση του σε πολλές εφαρμογές της. Ένα πολύ απλό παράδειγμα είναι το **Google Suggest** το οποίο εμφανίζεται κάθε φορά που πληκτρολογούμε κάτι για αναζήτηση στην μηχανή αναζήτησης της **Google**. Πιο πολύπλοκη και εκτενής χρήση του **AJAX** συναντάμε στην πλατφόρμα για ανταλλαγή ηλεκτρονικών μηνυμάτων της **Google**, το **Gmail**, ενώ μια από τις πρώτες εφαρμογές που έκανε γνωστή την τεχνολογία στο κοινό και παρουσίαζε με επιτυχία τις δυνατότητές της ήταν το **Google Maps**.

Μια από της πρώτες ολοκληρωμένες εφαρμογές που έκανε πλήρη χρήση και εκμετάλλευση του **AJAX** ήταν το **AJAX** Windows από την **Ajax13**. Ήταν ένα **διαδικτυακό Λειτουργικό Σύστημα** (**WebOS**) κατασκευασμένο πλήρως με **AJAX** τεχνικές. Ένα πιο σύγχρονο παράδειγμα είναι ο ιστότοπος **eBuddy** από την ομώνυμη εταιρία, όπου είναι κατασκευασμένη μια πλήρης εφαρμογή ενός διαδικτυακού προγράμματος ανταλλαγής μηνυμάτων (**web messenger**). Τέλος, ένα παράδειγμα χρήσης του **AJAX** σε διαδικτυακή εφαρμογή είναι το **Stripe Generator 2.0** από την **PopMinds**, όπου οι χρήστες μπορούν να δημιουργήσουν δυναμικά, μια ριγέ ταπετσαρία για την ιστοσελίδα τους. Παρακάτω θα παρουσιαστούν συνοπτικά οι προαναφερθείσες εφαρμογές.

8.1 Google Suggest



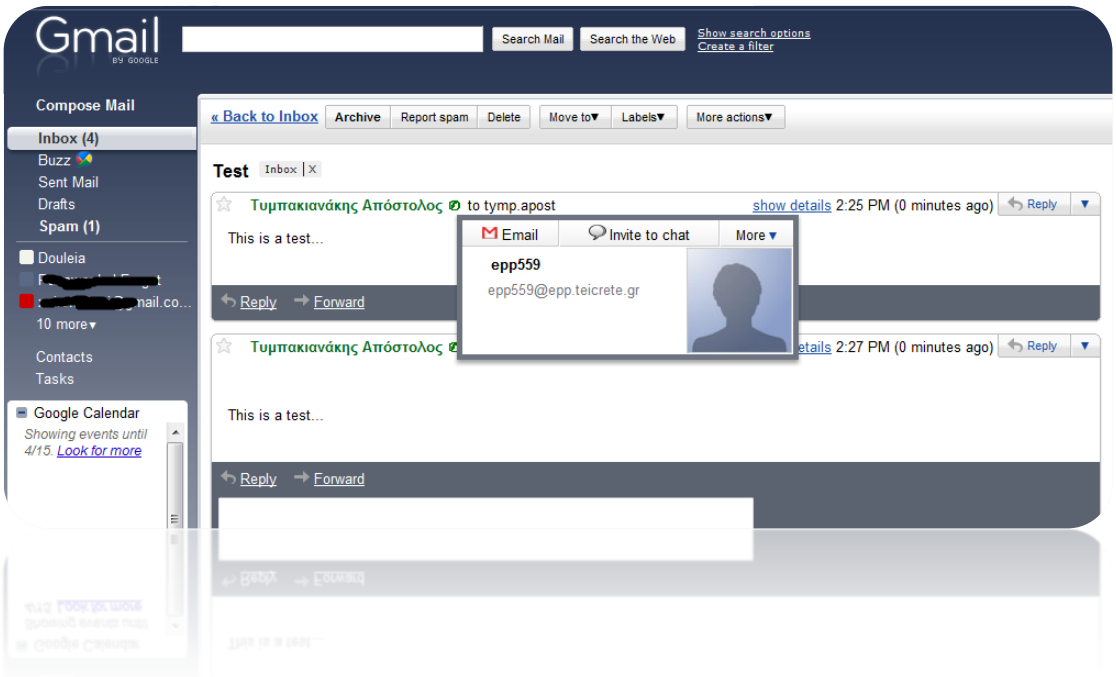
Το **Google Suggest** είναι ένα πολύ απλό παράδειγμα, του πόσο η εφαρμογή του **AJAX** στις υπάρχουσες εφαρμογές, βοήθησε στον εμπλουτισμό της λειτουργικότητας και της αμεσότητάς τους. Βρίσκεται ενσωματωμένο στην μηχανή αναζήτησης της **Google** (http://www.google.com). Το **Suggest** δουλεύει στο παρασκήνιο και, όταν ο χρήστης αρχίζει να πληκτρολογεί την λέξη-κλειδί (**keyword**) που θέλει να αναζητήσει, με ασύγχρονες αιτήσεις προς τους Διακομιστές της **Google** κάνει μια «πρόχειρη» αναζήτηση και του επιστρέφει ως απάντηση τις λέξεις-κλειδιά των δέκα πιο συχνών αναζητήσεων που περιέχουν αυτό που έχει πληκτρολογηθεί. Η διαδικασία αυτή εκτελείται δυναμικά, καθώς ο χρήστης πληκτρολογεί, και η εκτέλεσή της αρχίζει από το πρώτο γράμμα.

Έτσι ο χρήστης, πριν ακόμα πληκτρολογήσει ολόκληρη την λέξη-κλειδί που έχει στο μυαλό του, υπάρχει μεγάλη πιθανότητα να του έχει προταθεί ήδη ολόκληρη φράση που περιγράφει ακριβώς αυτό που ψάχνει. Όλα αυτά, με βάση πάντα τα στατιστικά που συγκεντρώνει η **Google** συνεχώς από τις καθημερινές αναζητήσεις των χρηστών της μηχανής αναζήτησής της.

Με αυτήν την έξυπνη εφαρμογή, η **Google** κατάφερε να απαλλάξει τους χρήστες από χρονοβόρες πολλαπλές αναζητήσεις ψάχνοντας την σωστή λέξι-κλειδί που θα τους εμφανίσει τα επιθυμητά αποτελέσματα, αφού εξ' αρχής προτείνεται στον χρήστη μια πιο πλήρες φράση-κλειδί με αυτό που ψάχνει, με συνέπεια σε μία μόνο αναζήτηση να έχει μπροστά του τα αποτελέσματα που αναζητά. Με αυτόν τον τρόπο γλιτώνει χρόνο ο ίδιος ο χρήστης, και η εταιρία απαλλάσσεται από τον φόρτο των επιπλέον αναζητήσεων.

Όλα αυτά γίνονται με μια εφαρμογή η οποία ουσιαστικά εκτελείται και ενσωματώνεται στο κουτί αναζήτησης (**search box**) της μηχανής, αντικείμενο οικείο για τον χρήστη, στην ίδια γνώριμη σελίδα που χρησιμοποιούσε και πριν, χωρίς να του παρουσιάζεται κάτι «ξένο», και προπαντός χωρίς να χρειαστεί κάποια επιπλέον ενέργεια, όπως επαναφόρτωση της σελίδας ή το πάτημα ενός κουμπιού.

8.2 Google Gmail

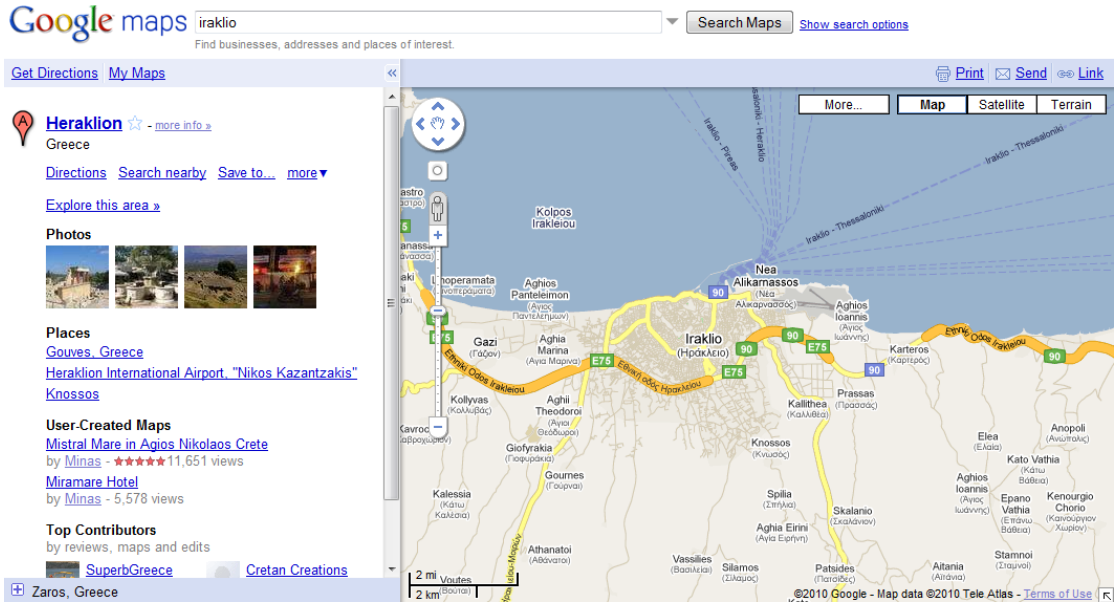


Το **Google Gmail** (http://mail.google.com) ήταν η πρώτη πλήρης εφαρμογή της εταιρίας που εκμεταλλευόταν πλήρως την τεχνολογία **AJAX** για να προσφέρει στον χρήστη μια πλήρης εφαρμογή διαχείρισης των ηλεκτρονικών μηνυμάτων του χωρίς τα μειονεκτήματα των άλλων παραδοσιακών εφαρμογών. Με το **Gmail** ο χρήστης «άνοιγε» τα εισερχόμενα μηνύματα του, έστελνε απαντήσεις ή τα προωθούσε σε φίλους του, τα έσβηνε ή τα αρχειοθετούσε, και όλα αυτά χωρίς να αλλάζει σελίδα, χωρίς επαναφορτώσεις και καθυστερήσεις.

Η **Google** εμπλούτισε το **Gmail** με πολλούς τρόπους από τότε που ξεκίνησε, έχοντας γίνει πλέον μια πλήρης πλατφόρμα γραφείου με ενσωματωμένες εφαρμογές για ηλεκτρονικό ημερολόγιο, λίστα εργασιών (**task-list**) και διαχείριση βιβλίου επαφών. Επίσης υπάρχει και εργαλείο που ενσωματώνει το **Google Docs** για διαχείριση και προβολή εγγράφων, καθώς και μια πλήρη εφαρμογή ανταλλαγής ηλεκτρονικών μηνυμάτων (**chat**) μεταξύ των χρηστών που έχουν λογαριασμό στο **Gmail**. Προχωρώντας ένα βήμα παραπέρα, η **Google** έχει εμπλουτίσει την πλατφόρμα με φωνητική συνομιλία και πρόσφατα και βίντεο-συνομιλία μεταξύ των χρηστών της.

Αυτό που είναι σημαντικό όμως είναι ότι όλες αυτές οι λειτουργίες-εφαρμογές που ενσωματώνονται σε αυτήν την πλατφόρμα λειτουργούν μέσα σε μία σελίδα, με χρήση της τεχνολογία **AJAX** για ασύγχρονη επικοινωνία με τον Διακομιστή, χωρίς να απαιτείται από τον χρήστη να περιμένει για επαναφορτώσεις ή αλλαγές σελίδας.

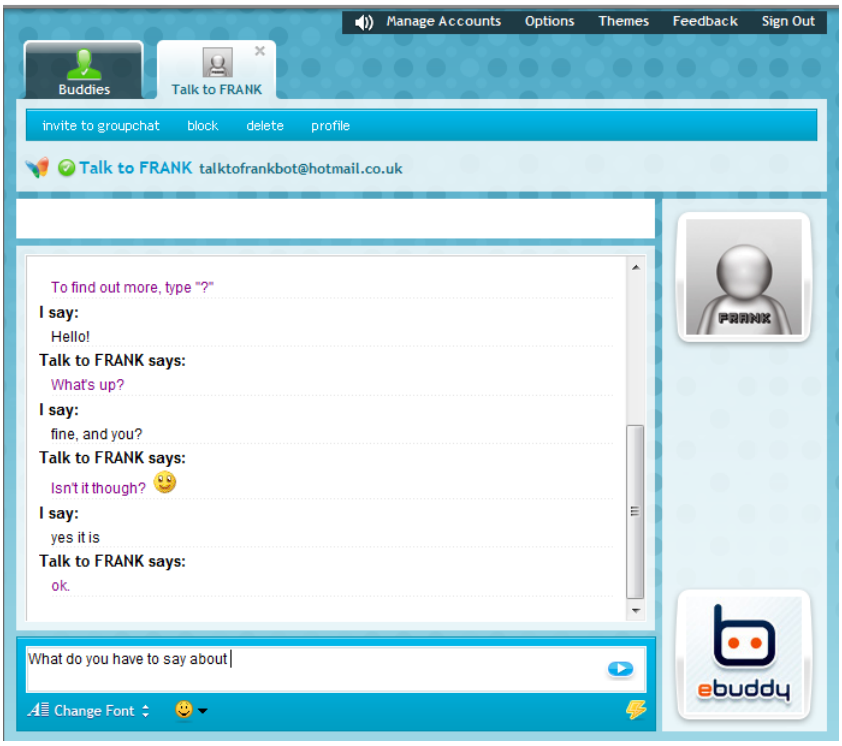
8.3 Google Maps



Το **Google Maps** (http://maps.google.com) ήταν η εφαρμογή που έκανε γνωστά στο ευρύ κοινό τα πλεονεκτήματα και τις δυνατότητες της χρήσης του **AJAX**. Η **Google** κατασκεύασε το 2005 μια εφαρμογή εξερεύνησης χαρτών διαφορετική από τις μέχρι τότε υλοποιήσεις. Έδινε την δυνατότητα στον χρήστη να περιηγείται στους χάρτες με τρόπο δυναμικό, δίνοντάς του την ψευδαίσθηση μιας συμβατικής εφαρμογής για σταθερό υπολογιστή. Ο χρήστης μπορούσε να κινηθεί προς όλες τις κατευθύνσεις με το ποντίκι του, εξερευνώντας τον χάρτη και πηγαίνοντας από σημείο σε σημείο με την ευκολία ενός κλικ. Μπορούσε να μεγεθύνει τον χάρτη και να δει περισσότερες λεπτομέρειες, σε βαθμό που έφτανε μέχρι και την εμφάνιση των παράδρομων μια πόλης. Παράλληλα εμφανίζονταν πληροφορίες για την περιοχή που είχε μπροστά του, πόλεις, χωριά, αξιοθέατα, σημεία ενδιαφέροντος, εικόνες, κτλ.

Όλα αυτά εκτελούνταν αδιάκοπα (**seamlessly**), με επικοινωνία στο παρασκήνιο χρησιμοποιώντας αιτήματα **AJAX** προς τον Διακομιστή, χωρίς να ενοχλείτε ο χρήστης, χωρίς να περιμένει για επαναφορτώσεις σελίδων και χωρίς να φεύγει από την κεντρική σελίδα της εφαρμογής που βρίσκονταν όλες οι πληροφορίες που τον ενδιέφεραν. Ο μόνος περιορισμός είναι η ταχύτητα της σύνδεσης του χρήστη.

8.4 eBuddy Web Messenger



Στην ιστοσελίδα του (http://web.ebuddy.com) το **eBuddy** προσφέρει στους χρήστες του την δυνατότητα να εισαχθούν σε έναν ή περισσότερους από τους λογαριασμούς τους στις υπηρεσίες που υποστηρίζει (**MSN Messenger**, **Yahoo IM**, **AOL IM**, **ICQ**, κ.α.) και να επικοινωνήσουν με τις επαφές τους σε πραγματικό χρόνο, χρησιμοποιώντας μόνο τον περιηγητή Ιστού (**Web Browser**) τους. Οι χρήστες, εκτός της ανταλλαγής μηνυμάτων, μπορούν ακόμη να διαχειριστούν τον λογαριασμό της εκάστοτε υπηρεσίας, να διαχειριστούν τις επαφές τους, να διαμορφώσουν το προφίλ τους, και άλλα.

Η περιήγηση στην λίστα των επαφών, η ανταλλαγή μηνυμάτων καθώς και όλη η εφαρμογή, έχει κατασκευαστεί έτσι ώστε να θυμίζει τις αντίστοιχες «**Desktop**» εφαρμογές, καταφέρνοντας να μιμηθεί την αμεσότητα και την ευκολία τους. Το **eBuddy** επίσης προσφέρει την επιλογή εμφάνισης της εφαρμογής σε 6 διαφορετικές γλώσσες, ενώ έχει κατασκευαστεί και ειδική έκδοση της εφαρμογής για χρήστες που την χρησιμοποιούν από το κινητό τους τηλέφωνο, και επομένως χρειάζονται μια έκδοση με χαμηλότερη χρήση εύρους ζώνης.

Και σε αυτήν την περίπτωση, η χρήση της ασύγχρονης επικοινωνίας επιτρέπει στην εφαρμογή να φτάνει σε ένα υψηλό επίπεδο ευχρηστίας, λειτουργικότητας και αμεσότητας, με τον χρήστη να απολαμβάνει τα οφέλη μιας πλήρους εφαρμογής ανταλλαγής μηνυμάτων από τον περιηγητή Ιστού του, χρησιμοποιώντας την όπου και αν βρίσκεται, από οποιοδήποτε

υπολογιστή ή συσκευή με πρόσβαση στο διαδικτύου και αν έχει στην διάθεσή του. Χωρίς μεγάλες καθυστερήσεις, περίπλοκες περιηγήσεις σε πολλαπλές σελίδες και επαναφορτώσεις.

8.5 Stripe Generator 2.0

Το **Stripe Generator 2.0** (http://www.stripegenerator.com) είναι ένα πολύ αντιπροσωπευτικό παράδειγμα μιας «μονοσέλιδης» εφαρμογής, όπως και οι «**Έξυπνες Επαφές**» αυτής της πτυχιακή εργασίας. Στην εφαρμογή που κατασκεύασε η **PopMinds**, παρέχεται δωρεάν στους σχεδιαστές διαδικτύου μια «γεννήτρια» παραγωγής ριγέ ταπετσαριών για το φόντο των σελίδων τους. Μια πολύ έξυπνη και χρήσιμη κατασκευή, η οποία επιτρέπει στον χρήστη, μέσα από αρκετές επιλογές και ρυθμίσεις, να δημιουργήσει το δικό του μοναδικό φόντο για την ιστοσελίδα του, ξεκινώντας ακόμα και από το μηδέν. Από τα χρώματα και τις φωτοσκιάσεις, μέχρι τον αριθμό, την πυκνότητα και το στυλ των ριγών, η εφαρμογή αυτή καταφέρνει να παρέχει μια πληθώρα χαρακτηριστικών, παραμένοντας όμως παράλληλα και πολύ απλή στην κατανόηση και την χρήση της. Επίσης, στην ίδια σελίδα προσφέρονται και οδηγίες χρήσης για την ενσωμάτωση των ταπετσαριών που δημιουργήθηκαν στις σελίδες των σχεδιαστών-χρηστών, με τις απαραίτητες ρυθμίσεις και επιλογές που απαιτούνται.

8.6 ajaxWindows

Το **ajaxWindows** (http://www.ajaxwindows.com/apps/windows/content/index.html) είναι ένα πρωτότυπο **διαδικτυακό Λειτουργικό Σύστημα** (**WebOS**), κατασκευασμένο πλήρως με τεχνικές **AJAX**, το οποίο βάζει υψηλά τον πήχη όσον αφορά τις δυνατότητες των διαδικτυακών εφαρμογών. Η κατασκευάστρια εταιρία, **Ajax13 Inc.** , προσπάθησε να μεταφέρει ένα μέρος του λειτουργικού συστήματος των **Windows** σε διαδικτυακό περιβάλλον, συνθέτοντας μια διαδικτυακή επιφάνεια εργασίας που τα έχει σχεδόν όλα. Ο χρήστης έχει στην διάθεσή του πολλών ειδών εφαρμογές, που ανάμεσά τους περιλαμβάνονται εφαρμογές γραφείου, όπως ο κειμενογράφος **ajaxWrite**, εφαρμογές πολυμέσων όπως το πρόγραμμα αναπαραγωγής μουσικής **ajaxTunes**, μίνι βοηθητικές εφαρμογές για την επιφάνια εργασίας (**Widgets**), ακόμα και μια βασική έκδοση ενός Πίνακα Ελέγχου υπάρχει και προσφέρει στον χρήστη μερικές απλές ρυθμίσεις του συστήματος.

Παρόλο που η υλοποίηση αυτή απέχει από το τέλειο, έχοντας κάποια προβλήματα, όπως ασυμβατότητας με κάποιους περιηγητές Ιστού ή και θέματα συνδεσιμότητας, δεν παύει να είναι μια πολύ αξιόλογη προσπάθεια. Συνδυάζοντας συνήθεις και απλές εφαρμογές, προσπαθεί και καταφέρνει να δείξει την δύναμη μιας πλατφόρμας εφαρμογών **AJAX**, καταφέρνοντας να ενσωματώσει ένα ολόκληρο λειτουργικό σύστημα σε μία ιστοσελίδα.

9

Βιβλιογραφία

[AJAX]
- ΧΟΛΖΝΕΡ ΣΤΗΒΕΝ (2009). ΟΔΗΓΟΣ ΤΗΣ AJAX. ΓΚΙΟΥΡΔΑΣ Μ.
- WIKIPEDIA (2009). Ajax (programming) http://en.wikipedia.org/wiki/Ajax_%28programming%29
- Jesse James Garrett (2005). Ajax: A New Approach to Web Applications http://www.adaptivepath.com/ideas/essays/archives/000385.php
- W3schools.com (2009). AJAX Tutorial http://www.w3schools.com/Ajax/Default.Asp

[jQuery]
- jQuery (2009). jQuery Documentation http://docs.jquery.com/Main_Page
- WIKIPEDIA (2009). jQuery http://en.wikipedia.org/wiki/JQuery

[ColdFusion]
- Adobe (2009). ColdFusion Developer Center http://www.adobe.com/devnet/coldfusion/
- Adobe (2009). Developing Adobe ColdFusion 9 Applications http://help.adobe.com/en_US/ColdFusion/9.0/Developing/index.html
- WIKIPEDIA (2009). ColdFusion http://en.wikipedia.org/wiki/ColdFusion

[JSON]
- WIKIPEDIA (2009). JSON http://en.wikipedia.org/wiki/JSON
- JSON.org (2009). Introducing JSON http://www.json.org/

[DOM]
- WIKIPEDIA (2009). Document Object Model http://en.wikipedia.org/wiki/Document_Object_Model
- W3C (2009). Document Object Model (DOM) http://www.w3.org/DOM/
- W3schools.org (2009). HTML DOM Tutorial http://www.w3schools.com/htmldom/default.asp

[MVC]
- WIKIPEDIA (2009). Model-view-controller http://en.wikipedia.org/wiki/Model-view-controller

[XMLHttpRequest]
- WIKIPEDIA (2009). XMLHttpRequest http://en.wikipedia.org/wiki/XMLHttpRequest
- W3schools.org (2009). The XMLHttpRequest Object http://www.w3schools.com/ajax/ajax_xmlhttprequest.asp

[WebService]
- WIKIPEDIA (2009). Web service http://en.wikipedia.org/wiki/Web_service
- W3schools (2009). Web Services Tutorial http://www.w3schools.com/webservices/default.asp

[ΓΕΝΙΚΑ]
- W3C.org (2009). W3C Standards http://www.w3.org/standards/